

برمجة الأبعاد الثنائية والثلاثية للكمبيوتر

باستخدام تكنولوجيا DirectX ومسرات الأبعاد الثنائية
والثلاثية على الأنظمة Windows 9x/2000 تعرف على أسرار
برمجة الصوت والصورة لجهاز الكمبيوتر، بما في ذلك
الألعاب الإلكترونية، برامج التصور الحقيقي، برامج
الأبعاد الثنائية والثلاثية وكل ما يستغل هذه التكنولوجيا.

استخدم المكتبة Genesis 3D لإنشاء عوالم ثلاثية الأبعاد
واستخدم المكتبة AGDX لإنتاج برامج Multimedia
باستخدام ملفات الصور والفيديو والملفات الصوتية.

يأتي مع الكتاب قرص مدمج يحتوي على جميع
الأمثلة المشروحة وكذلك البرامج المرافقة وبرامج
أخرى إضافية باستخدام قائمة عربية بالصوت
والصورة.



CD-ROM

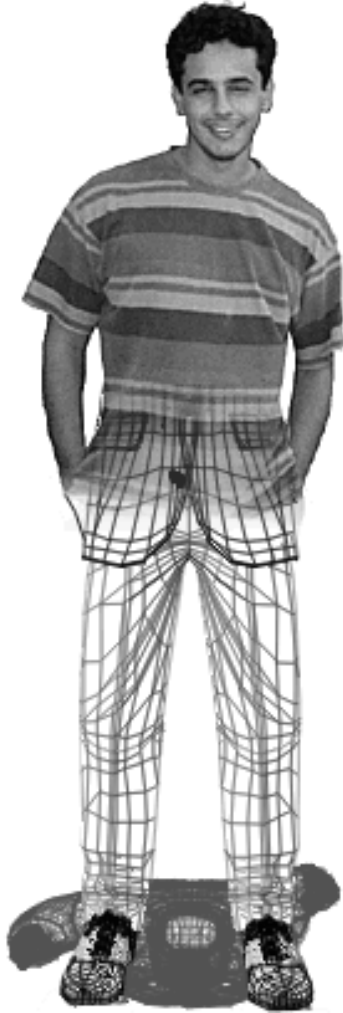


تأليف المهندس : عبدالناصر سيف الكعبي



برمجة الأبعاد الثنائية والثلاثية

للكمبيوتر



تأليف

تصميم الرسوم

برمجة

تصميم الغلاف الخارجي

عبدالناصر الكعبي

نعم ، هذا الكتاب هو عمل شخص واحد. من قال

أن النجوم لا تظال.

إهداء وشكر

هناك الكثيرين ممن أنا شاكر لهم مساعدتهم بشكل أو بآخر لجعل هذا الحلم حقيقية بين يدي كل قارئ. ولن أستطيع بطبيعة الحال ذكرهم جميعاً ولكن عملهم موجود في القلب وصنيعهم مشكور. أولاً بعد الشكر لله عز وجل أشكر بشكل خاص دولة الإمارات شعباً وقائداً. وخاصة الوالد الشيخ زايد الذي لم يبخل على أبنائه بالتعليم أو النصح أو التشجيع. ولسعادة أحمد بن خليفة السويدي الذي وفر لي الفرصة في أن يكون هذا الكتاب واقع ملموس.

ثم أهدي هذا الكتاب لروح جدتي مريم بنت سيف (1935 – 1996) رحمها الله والتي علمتني أنه لا يأس مع الحياة ولا حياة مع اليأس. وكذلك إلى روح عمي سعيد بن محمد رحمة الله الذي علمني أن ليس للعلم مكان ولا زمان وليس لعمل الخير نهاية. ثم أهدي هذا الكتاب إلى والدي الحبيبين والذين لم يبخلوا علي بالغالي والنفيس لتحويل الأحلام إلى واقع والضعف إلى قوة. ثم إهداء خاص لأم خالد التي تحملت إزعاجي لها الليالي الطويلة وأنا منهمك في البرمجة وكتابة هذا الكتاب. ولا أنسى أبني الصغير "خالد" الذي كان نبراسي في الليالي الطويلة، وبقية عائلتي وأساتذتي في الجامعة وأولاً وأخيراً القراء الكرام الذين تلقيت منهم الكثير من التشجيع عن طريق بريدي الإلكتروني، شكر لكم وجزاكم الله خيراً.

عبدالناصر بن سيف بن هلال بن خصيف الكعبي

الموضوع ❖	الصفحة	جدول المحتويات
	1	المقدمة
		<p>الجزء الأول</p> <p>الجزء الثاني</p> <p>الجزء الثالث</p>
	15	الجزء الأول
	17	الوظائف Functions ❖
	22	المتغيرات من نوع Struct ❖
	23	المتغيرات من نوع enum ❖
	24	الذاكرة ❖
	29	بطاقات AGP ❖
	30	حفظ المعلومات على القرص الصلب ❖
	42	تعطيل عمل حافظ الشاشة ❖
	44	هل تعلم؟ ❖
		الجزء الثاني
		الجزء الأول
	49	لماذا نتحدث عن تركيب الأدوات؟ ❖
	49	أولاً ما هي تكنولوجيا DirectX؟ ❖

- 50 ❖ تركيب تكنولوجيا DirectX
- 57 ❖ على ماذا تحتوي تكنولوجيا DirectX ؟
- 59 ❖ ما هي بيئة التطوير Visual C++ ؟
- 60 ❖ تركيب بيئة التطوير Visual C++
- 66 ❖ هل نحن جاهزون لاستخدام بيئة التطوير ؟
- 70 ❖ هل هذا هو كل شيء ؟
- 75 ❖ تركيب المكتبة AGDX

مركز الأبحاث
العلمية والتقنية
في الرياضيات
والعلوم التطبيقية
والهندسة
والطب
والصيدلة
والبيولوجيا
والصيدلانية
والصيدلانية

- 81 ❖ win32 برمجة win32
- 95 ❖ كيف تضيف Resources لبرنامج win32
- 98 ❖ Win32 Application

مركز الأبحاث
العلمية والتقنية
في الرياضيات
والعلوم التطبيقية
والهندسة
والطب
والصيدلة
والبيولوجيا
والصيدلانية
والصيدلانية

- 107 ❖ Tiles
- 107 ❖ بعض الإحصائيات
- 108 ❖ مرتبط الفرس
- 109 ❖ ما هي المعضلة ؟

- 109 ❖ ما هو الحل إذا ؟
- 111 ❖ كيف ننشئ التايلز
- 112 ❖ برنامج MapMaker
- 113 ❖ Autograb.exe
- ❖ طريقة إنشاء خلفية متحركة لصور
- 116 ❖ التايلز باستخدام برنامج MapMaker

الفصل الثاني
 التايلز

AGDX المكتبة

- 123 ❖ المكتبة AGDX
- 123 ❖ قصة المكتبة AGDX و CDX
- 124 ❖ المميزات
- 128 ❖ كيف نستخدم المكتبة AGDX في برامجنا

الفصل الثالث
 التايلز

AGDX (1-15) أمثلة

- 132 ❖ أمثلة المكتبة AGDX
- 135 ❖ المثال الأول - صورة على الشاشة - كاملة -
- 145 ❖ المثال الثاني - صورة على الشاشة - نافذة -
- 149 ❖ أمر رسالة التوقيت

- ❖ المثال الثالث – لتتخلص من win32 وبرمجتها 153
- ❖ المثال الرابع – صوت ، صورة ، موسيقى و.. اكشن 161
- ❖ المثال الخامس – الخلفيات التطبيقية – التحكم في الإدخال 169
- ❖ المثال السادس – نريد أن نرى الحركة ! 175
- ❖ منسق الإطارات Frame Tiler 180
- ❖ مفتاح الألوان Color Key 181
- ❖ المثال السابع – أين هي خلفيات التايلز ؟ 190
- ❖ المثال الثامن – النوع الثاني من خلفيات التايلز ؟ 198
- ❖ المثال التاسع – الخلفيات المستطيلة (باناراميك) 206
- ❖ المثال العاشر – أين هي خلفيات التايلز ؟ مرة أخرى 216
- ❖ المثال الحادي عشر – الخلفيات التطبيقية 226
- ❖ المثال الثاني عشر – الملفات السينمائية 238
- ❖ المثال الثالث عشر – الطبقات شبه الشفافة 238
- ❖ وملفات برنامج PhotoShop 248
- ❖ المثال الرابع عشر – نوع آخر من الممثلين وطرق الإدخال (لوحة المفاتيح – إشارة الفأرة – عصا الألعاب) 262
- ❖ المثال الخامس عشر – نريد أن نبرمج برنامج كاملا ونبيعه في الأسواق 284
- ❖ الآلة الوضعية (State machine) 287

❖ هل نستخدم 256 لون فقط أم آلاف الألوان ؟

295

الفصل الرابع

البرمجة والرسوم المتحركة

304

309

❖ نظام بولر

311

❖ الحركة المفصلة

318

❖ من ماذا يتكون كائن الحركة ؟

الفصل الخامس

330

Bryce

332

❖ Bryce3D

335

❖ إنشاء Create

344

❖ تعديل Edit

345

❖ السماء والضباب Sky &Fog

351

❖ بعض المميزات الجديدة في الإصدار الرابع

الفصل السادس

Poser

354

356

❖ الحركة وبرنامج Poser

358	❖ الشاشة الرئيسية
360	❖ لماذا نستخدم فلم متحرك كخلفية ؟
362	❖ لماذا نحتاج أن نختار أي جسم ؟
363	❖ أدوات وأسطوانات التحكم بالجسم
365	❖ النقاط
366	❖ أدوات التحكم بالكاميرا
367	❖ مستطيل المكتبات
369	❖ الحركة
371	❖ القائمة العلوية
372	❖ الخلاصة

الفصل الثاني
البيئات الافتراضية

374	Genesis 3D
375	❖ 2D & 3D
380	❖ Genesis 3D
382	❖ خطوات إنشاء برنامج عن طريق مكتبة Genesis 3D
384	❖ تركيب مكتبة Genesis 3D
389	❖ المصمم — برنامج تصميم عوالم الأبعاد الثلاثية World Editor
397	❖ مثال لاستخدام المكتبة Genesis 3D

420

Linked Lists **القوائم المتصلة**

422

❖ الحجز المتغير للذاكرة

423

❖ ما هي القوائم المتصلة ؟

425

❖ إضافة وإزالة نقطة اتصال من القائمة المتصلة

434

❖ بعض الإضافات

438

DirectDraw **العنصر**

440

❖ DirectDraw العنصر

441

❖ ما هي DirectDraw ؟

451

❖ العنصر DirectDraw- المثال الثاني

457

❖ السطح الرئيسي والسطح الخلفي

464

AGDX **البنية الرسومية**

466	البنية الداخلية لـ AGDX	❖
468	Surface Class	❖
471	Screen Class	❖
480	البرنامج	❖
483	ملفات الصور مستقلة النوعية	❖

الفصل الرابع عشر
تكنولوجيا DirectX

486	Direct3D	❖
488	البعد الثالث	❖
492	مصدر الإضاءة	❖
493	Direct3D	❖
494	Retained Mode	❖
495	Immediate Mode	❖
497	Direct3DRM الواجهة الرئيسية	❖
499	تعديل مسار البحث	❖
501	DIRECT3DRMDEVICE واجهة الأداة	❖
503	Rendering أنواع الـ	❖
505	DIRECT3DWINDEVICE الواجهة البديلة	❖
506	DIRECT3DRMVIEWPORT واجهة وجهة النظر	❖
507	CLIPPING أو الحذف	❖
507	FRAME INTERFACE واجهة الإطار	❖
509	موقع الإطارات	❖

- 511 DIRECT3DRMMESHBUILDER واجهة بنية الأجسام ❖
- 512 Rendering أنواع الـ ❖
- 514 التحكم في الأوجه ❖
- 514 التحكم بالنقاط (vertex) ❖
- 515 الحركة والحجم ❖
- 515 سرعة الأداء ❖
- 516 DIRECT3DRMMESH واجهة الأجسام ❖
- 517 التحكم بالنقاط (vertex) ❖
- 517 إنشاء جسم ثلاثي الأبعاد عن طريق استخدام واجهة بنية الأجسام ❖
- 518 DIRECT3DRMFACE واجهة الأوجه المستخدمة ❖
- 518 أوامر خريطة الشكل الخارجي للوجه ❖
- 519 أوامر إضاءة الشكل الخارجي للوجه ❖
- 519 أوامر نقاط الأوجه ❖
- 520 DIRECT3DRMTEXTURES واجهة خريطة الشكل الخارجي ❖
- 520 صنع الشكل الخارجي ❖
- 521 DIRECT3DRMTEXTUREWRAP واجهة الغلاف الخارجي للأجسام ❖
- 522 DIRECT3DRMMATERIAL واجهة الإضاءة للشكل الخارجي ❖
- 523 قوة لمعان الإضاءة ❖
- 523 لون لمعان الإضاءة ❖
- 524 لون الإضاءة المنبعثة ❖
- 524 DIRECT3DRMLIGHT واجهة الإضاءة ❖

- 548 ❖ تعبئة الأجسام ثلاثية الأبعاد
- 550 ❖ مثال لبرنامج **Direct3D Retained Mode**
- 551 ❖ كيفية بداية المثال وما يقوم به
- 553 ❖ إنشاء تعاريف المتغيرات المستخدمة
- 555 ❖ إعداد وإنشاء نافذة على الشاشة
- 557 ❖ وظيفة **InitApp**
- 561 ❖ عملية إنشاء النافذة الرئيسية
- 563 ❖ إعداد الأدوات **Enumerating Device Drivers**
- 563 ❖ الأمر **EnumDrivers** (أمر الإعداد)
- 565 ❖ الأمر **enumDeviceFunc**
- 567 ❖ الأمر المساعد **BPPToDDBD**
- 568 ❖ إعداد محيط ثلاثي الأبعاد
- 569 ❖ إنشاء الأداة ووجهة النظر
- 571 ❖ وضع مستوى الشكل الخارجي للبرنامج (**Render State**)
- 572 ❖ إنشاء دورة الإخراج (**Rendering Loop**)
- 574 ❖ إنشاء المشهد (**Creating the Scene**)
- 575 ❖ وظيفة **My Scene**
- 577 ❖ وظيفة **MakeMyLights**
- 578 ❖ وظيفة **SetMyPositions**
- 579 ❖ وظيفة **MakeMyMesh**
- 580 ❖ وظيفة **MakeMyWrap**

- 582 AddMyTexture وظيفة ❖
584 Cleaning Up إنهاء البرنامج ❖

الفصل الثامن عشر
البرمجة باستخدام AGDX

- 586 AGDX ❖
588 مرجع أوامر المكتبة AGDX ❖
589 AGDXScreen ❖
596 AGDXSurface ❖
604 AGDXLayer ❖
607 AGDXTile ❖
609 AGDXMap ❖
614 AGDXSprite ❖
619 AGDXSpriteList ❖
622 AGDXInput ❖
625 AGDXMusic ❖
627 AGDXSound ❖
629 AGDXSoundBuffer ❖
632 Global functions ❖
634 كيف تستطيع أن تجدني
636 المراجع

المقدمة

الحمد لله رب العالمين ، والصلاة والسلام على اشرف المرسلين سيدنا محمد وعلى آله وصحبه أجمعين. أما بعد ، عزيزي القارئ مرحبا بك في عالم جديد بلا حدود أو حواجز ، ليس بالواقع ولكنه واقع ، عالم يعيش في ذاكرة الكمبيوتر ، تلك الرقائق السلكونية التي تحوي في داخلها الجبال والبحار ، الجريمة والشجاعة ، الحب والكراهة. ما بيننا وبين هذا العالم الثلاثي الأبعاد مجرد شاشة ثنائية تفصلنا عنه ، ولكن هل هذا هو كل ما في الأمر ؟ التكنولوجيا تتطور بشكل يومي ، ولا تستغرب أن ترى يوما ما تظنه مجرد برنامج على الكمبيوتر يتحول إلى عالم أنت تعيشه وتتأثر به. هنا يتبين الخيط الرفيع بين الواقع والخيال بين الحقيقية والتصوير. هل يمكننا أن نعيد إحياء الأموات ؟ هل نستطيع أن نعيد التاريخ من جديد ؟ هل يمكننا أن نحول ما في عقولنا من تخيلات إلى واقع ملموس ؟ الإجابة نعم نستطيع ولكن عن طريق غير الذي كنا نتصوره. وما هذه إلا بدايته. فمرحبا بك معنا في هذه الرحلة الشيقة الشبيهة برحلة من رحلات السندباد.

سوف نتطرق في هذا الكتاب لموضوع برمجة الأبعاد الثنائية والثلاثية للكمبيوتر. وهدفنا هو إعطائك القدرة ليس للتعرف فقط على كيفية برمجة تكنولوجيا DirectX بل كذلك استخدام مكتبته خاصة (AGDX) ومجانا مع هذا الكتاب بما في ذلك الشفرة البرمجية لها. هذه التكنولوجيا التي غيرت طريقة التعامل مع عالم الكمبيوتر من اللحظة التي خرجت فيها إلى العالم. تعلم عن طريقها كيف تستطيع أن تجعل من الصور الصامتة الميتة شي حي على شاشة الحاسب.

بالإضافة إلى ذلك سوف نتعرف على المكتبة الثلاثية الأبعاد Genesis 3D وسنقوم عن طريقها بالتعرف على كيفية سهولة برمجة الأبعاد الثلاثية ثم نتعرف على بعض برامج التصميم ثلاثية الأبعاد (Bryce 3D&4 و Poser 3) ونستخدمها في أمثلتنا. وأخيرا سوف ننظر إلى العنصر Direct3D أحد عناصر تكنولوجيا DirectX والمهتم بالأبعاد الثلاثية , وذلك بشكل مفصل و ثم نتعرف على جميع أوامره وكيف نستطيع الاستفادة منها. الشيء الطريف أنني كتبت هذه المقدمة عند انتهائي من الكتاب وليس في بدايته ومع ذلك سميتها المقدمة. ومما يستحق ذكره , أنه حتى المكتبات الغربية تفتقر لهذه النوعية من الكتب ولإيماني أن الهدف من هذا الكتاب هو رفع المستوى العلمي للحاسب الآلي في وطننا العربي لم أتردد للحظة في أن يكون هذا الكتاب باللغة العربية للقارئ العربي.

عزيزي القارئ قد يتبادر إلى أذهاننا السؤال التالي : ماذا نقصد ببرمجة الأبعاد الثنائية والثلاثية للكمبيوتر ؟ طبعاً نحن نسمع كثيرا عن هذه المصطلحات ولكننا في هذا الكتاب سنقوم بالنظر إليها من الناحيتين : البرمجية و التصميمية. بمعنى أننا سوف نتعرف على كيفية تصميم الصور الثنائية والثلاثية الأبعاد ثم نقوم ببرمجتها واستخدامها في برامجنا.

إلى ماذا ينقسم هذا الكتاب ؟

ينقسم هذا الكتاب إلى قسمين ، في القسم الأول نتعامل مع تكنولوجيا DirectX عن طريق مكتبة خاصة تقوم باستغلال جميع إمكانيات هذه التكنولوجيا مع سهولة الاستخدام. وسوف ننظر كذلك إلى مكتبة برمجية مهتمة ببرمجة الأبعاد الثلاثية ثم ننشئ مجموعة من الأمثلة للإبعاد الثنائية والثلاثية ، أما في القسم

الثاني فإننا سنتعرف على طريقة عمل تكنولوجيا DirectX بشكل مباشر وبدون استخدام مكتبة معينة وبالأخص العنصر Direct3D وذلك بحيث يستطيع القارئ معرفة كيفية عمل هذه التكنولوجيا بالإضافة إلى كيفية البرمجة بها بشكل مباشر. وكذلك مدى فعالية المكتبة المرفقة في القسم الأول في جعل استخدام DirectX عملاً سهلاً وممتعاً.

في كتابي الأول "برمجة ألعاب الكمبيوتر على النظام windows95"



فقد استخدمت مكتبة خاصة للألعاب والتي عن طريقها يستطيع المبرمج استخدام تكنولوجيا DirectX بدون القلق بطريقة عملها وقد صممت تلك المكتبة بشكل خاص للألعاب وقد كانت ردود الفعل إيجابية حيث أن الكثير من القراء لا يهتم بمعرفة تفاصيل عمل تكنولوجيا DirectX مادام

باستطاعته استغلالها في برامجه كما هو الحال مع استخدام MFC (Microsoft Foundation Classes) وحيث أنه لا أحد يهتم بطريقة برمجتها مادامنا نستطيع أن نستغلها في برامجنا. ولكن أحياناً نرى الحاجة إلى معرفة برمجة تكنولوجيا DirectX بشكل مباشر. مثلاً لو أردنا إضافة أو تغيير شيء معين في مكتبتنا ، أو إذا كان القارئ يهتم بالطريقة التي تمت بها برمجة المكتبة المستخدمة فإننا قد خصصنا قسم خاص في هذا الكتاب يقوم بشرح ذلك. وباستخدام المكتبة المرفقة مع هذا الكتاب (AGDX) فإننا سوف نملك القدرة على إنشاء برامج الميديا كلها بشكل عام بما في ذلك الألعاب (سوف نقوم باستخدام الكثير من الأمثلة لشرح ذلك) ثم نتطرق إلى استخدام الأبعاد الثلاثية وكيفية برمجتها ، لاحظ بأن هدف الكتاب هو إعطاء القارئ القدرة على القيام ببرمجة الأبعاد الثنائية والثلاثية وذلك بأبسط طريقة ممكنة سواء تحت النظام Windows95 أو Windows98 أو Windows 2000. و

يجب معرفة أنه عند القيام ببرمجة برنامج معين سواء كان لعبة مغامرات أو برنامج تعليمي يستخدم الصوت والصورة وحتى بعض مقاطع الفيديو أو برامج التصوير الحقيقي (Virtual reality) فإن الوقت الذي يستغرق لإنهاء ذلك البرنامج لا يكون بالضرورة في الشفرة البرمجية للبرنامج ولكن في كثير من الأحيان يكون في إعداد الصور والأصوات المناسبة لاستخدامها في البرنامج ، لذلك سوف ننظر إلى بعض البرامج والأدوات لمساعدتنا بالقيام بهذه المهمة.

ما هي تكنولوجيا DirectX ؟

هي عبارة عن مجموعة أوامر تعطينا القدرة على التحكم في ذاكرة الكمبيوتر بشكل أكبر وأسهل بحيث تسهل عملية برمجة الصوت والصورة على شاشة الكمبيوتر وسوف نقوم بشرح طريقة عمل وكيفية استخدام أهم العناصر في هذه التكنولوجيا وذلك بشكل مفصل وواضح.

كم يجب أن يكون مدى معرفتي بلغة السي المحسنة C++ ؟

إذا كانت معرفتك بلغة السي (C) أو السي المحسنة (C++) جيدة فإنه سوف يعجبك هذا الكتاب لأنه يستخدم هاتين اللغتين في كل برامج الكتاب و إذا كنت لا تعرف لغة السي (C) أو السي المحسنة (C++) فإنك ستكون مستغرباً من مدى سهولة البرمجة تحت النظام Windows وذلك باستخدام المكتبة المرفقة. (لاحظ أن السي المحسنة (C++) هي لغة مطورة من لغة السي (C) وتشمل كل أوامرها مع بعض الإضافات التي يجب ألا نشغل أنفسنا بها)

ما هي اللغة أو بيئة التطوير المستخدمة في هذا الكتاب ؟

بيئة العمل المستخدمة في هذا الكتاب هي Visual C++ تحت النظام Windows95/98/2000 ونستخدم الإصدار السادس 6 Visual C++ أو أعلى هنا لسهولة متابعة خطوات إنشاء الأمثلة في هذا الكتاب وهذه هي البيئة المستخدمة من قبل الكثيرين من مبرمجي DirectX. وتحتوي على برنامجين مهمين "المصرف" (Compiler) و "الموصل" (Linker) والذين عن طريقهما نقوم بإنتاج برامجنا في هذا الكتاب.



هل يفترض الكتاب معرفة القارئ للغة السي الحسنة (C++) ؟

موضوع هذا الكتاب ليس للتعريف بلغة السي المحسنة لأن الكتب المختصة في هذه اللغة كثيرة، ولكن سيتم استخدامها كأداة. ومعرفة هذه اللغة بالذات ليس ضرورياً ولكن يُفترض معرفة القاري للأساسيات بأحد لغات الكمبيوتر.

هل بيئة التطوير موجودة في القرص المدمج (CD-ROM) لهذا الكتاب ؟

لا، كنت أود أن أجعلها من ضمن البرامج الموجودة في القرص المدمج ولكن لو فعلت ذلك فإن Microsoft سوف تطاردني في الألف سنة المقبلة وحتى لو قبلت المطاردة و جعلت بيئة التطوير من ضمن البرامج فإن الحيز الذي سوف تشغله لن يعطي الفرصة لإضافة البرامج الأخرى القيمة والمتضمنة مع القرص المدمج.

لماذا لم تختار بيئة تطوير غير Visual C++ ؟

طبعا هناك اكثر من بيئة تطوير متوفرة للغة السي المحسنة بل أن بعضها مجاني وكنت أود لو أستطيع إرفاق بيئة التطوير مع هذا الكتاب حتى لا يحتاج القارئ لشرائها بشكل منفصل. ولكن الحقيقة أن كل من ينظر إلى برمجة الحاسب الآلي بشكل جدي وباستخدام نظام النوافذ فإنه سيقوم بشراء بيئة Visual C++ لتمتعها بمميزات تكاد تكون الوحيدة من نوعها ويجب أن لا ننسى أن شركة مايكروسوفت هي المنتجة لهذه البيئة ولتكنولوجيا DirectX وكذلك لنظام التشغيل Windows.

هل تكنولوجيا DirectX موجودة في القرص المدمج (CD-ROM) لهذا الكتاب ؟

نعم موجودة في القرص المدمج لهذا الكتاب لأن شركة مايكروسوفت تعطي هذه التكنولوجيا مجانا على صفحاتهم على الإنترنت وقد أرفقناها مع هذا الكتاب. انظر للفصل الثاني للتعرف على كيفية تركيبها وكذلك تركيب بيئة التطوير.

هذا الكتاب موجه لمن ؟

هذا الكتاب موجه لكل شخص يريد أن يتعرف على التكنولوجيا المستخدمة لإنشاء برامج الصوت والصورة والتي تحاول الشركات البرمجية إخفائها عن العامة. الآن تستطيع أنت كشخص واحد القيام ببرمجة جميع البرامج التي تستخدم المؤثرات الصوتية والمرئية وحتى الحسية. وسوف نضع بعض الاهتمام على برمجة الألعاب لأنها تشمل جميع الصفات التي قد توجد في برامج الملتي ميديا (Multimedia) وهي برامج الصوت والصورة و برامج التصوير الحقيقي وبرامج الطيران وكل ماله علاقة بالحركة والصوت على شاشة الكمبيوتر. ولكن طبعا مبرمجي الألعاب سوف يجدون الكثير من المعلومات عن برمجة الألعاب الثنائية والثلاثية الأبعاد.

ماذا عن سرعات 3D/FX ثلاثية الأبعاد ؟

عندما نتكلم عن الأبعاد الثلاثية فإن Direct3D هي محل اهتمامنا ولكن تجاهل السرعات المستخدمة لمعالجات Voodoo 1/2/3 و Voodoo Rush هي نوع من تجاهل مستقبل الأبعاد الثلاثية على الكمبيوتر ومن المعروف أن هذا النوع من السرعات يعطينا نفس جودة أجهزة الألعاب التجارية ذات الأبعاد الثلاثية. وإن كان القارئ لا يملك فكرة عن هذه السرعات فإنها صارت المنافس القوي لتكنولوجيا DirectX. سوف نقوم باستخدام



مكتبة Genesis3D لإنتاج بعض الأمثلة والتي بدورها سوف تقوم باستخدام جميع أنواع هذه التكنولوجيا.

هل مكتبة AGDX موجودة في القرص المدمج (CD-ROM) لهذا الكتاب ؟

نعم ، المكتبة AGDX طورت خصيصا لهذا الكتاب لتسهيل برمجة تكنولوجيا DirectX وتأتي مع الشفرة البرمجية كاملة، وهي أيضا موجودة من ضمن موقع الألعاب العربية <http://www.ArabGames.com> .

وهل مكتبة Genesis 3D موجودة في القرص المدمج (CD-ROM) لهذا الكتاب ؟

نعم موجودة ، لأن شركة Eclipse Entertainment تعطي هذه التكنولوجيا مجانا على صفحاتهم ونحن أرفقناها مع الكتاب بكل ملحقاتها.

عن ماذا يتحدث هذا الكتاب وعن ماذا لا يتحدث ؟

الموضوع الرئيسي لهذا الكتاب هو برمجة الأبعاد الثنائية والثلاثية. وسوف نتطلع على افضل الأساليب والأدوات التكنولوجية المتوفرة لدينا لأداء هذا الغرض ، مع الإبقاء على سهولة القيام بهذه المهمة وذلك على أكمل وجه. وليس هدف هذا الكتاب تعليم لغة السي المحسنة (C++) ولكننا سننظر إلى أهم أوامرها المستخدمة في أمثلة هذا الكتاب. وبشكل عام سوف نفترض أن القارئ مبتدئ في هذه اللغة وننظر إليها كأداة لأداء الغرض المطلوب لا أكثر. وللتعمق أكثر في هذه اللغة الرائعة أنصحك بالإطلاع على كتبها الكثيرة المتوفرة في الأسواق.

ما هي مميزات الحاسب الآلي الذي سوف نستخدمه في هذا الكتاب ؟



- معالج Pentium I/II/III أو متوافق
- 32 ميجا بايت أو أكثر من ذاكرة النظام
- بطاقة فيديو (يستحسن استخدام بطاقة 3dfx)
- نظام Windows95 أو Windows98 أو Windows2000

ماذا عن الشفرة المكتوبة في هذا الكتاب ، هل سأقوم بكتابتها مرة أخرى كلما حاولت تجربة مثال ؟ هذا قد يأخذ مني ساعات ، غير الأخطاء التي يمكن أن تحصل .

لا تقلق أنا أيضا يدي تؤلمني من جراء كتابة الشفرة البرمجية في كتب الكمبيوتر التي بحوزتي. في هذا الكتاب جميع الشفرة البرمجية التي تراها ستجدها في القرص المدمج الملحق مع الكتاب ولن تحتاج لطباعة سطر واحد. بل انك تستطيع تجربته وتشغيل الأمثلة والدروس مباشرة حتى قبل أن تقرأها. بهذه الطريقة تحصل على فكرة واضحة عما يتكلم عنه الفصل أو المثال.

وأخيراً.....

الكتاب ، أي كتاب كان ، ما هو إلا حبر على ورق وليست له قيمة سوى تزكيته للنار في ليلة شتاء باردة ليعطينا بعض الدفء. ولكن هل هذه فعلا هي قيمة الكتاب ؟ مجرد مجموعة أوراق ؟ طبعاً لا لأن القيمة الحقيقية للكتاب تكمن في معنى الكلمات التي كتبت فيه. وليس من فائدة ترجى لكتاب ملقي في ركن من أركان مكتبتك وليس هناك من يقرأه. لقد اكتشفت من تجربتي السابقة في الكتابة أن أكثر علامات الاستفهام تظهر لأن القارئ يقفز من جزء معين للكتاب إلى جزء آخر بدون فهم واستيعاب. ومع أنني حاولت جاهداً فصل كل موضوع عن الآخر بحيث يبقى مستقلاً إلا أن ذلك يبدو مستحيلاً أحياناً. لذا أرجو منك عزيزي القاري أن تبدأ في قراءة الكتاب حسب الترتيب الذي وضع عليه. وذلك لأنني عندما أتكلم عن موضوع معين افترض أحياناً أن القارئ سبق وقراء الموضوع الذي قبله. هذا الكتاب ليس مشروباً سحرياً يجعلك تفهم ما يحويه ، ولكن مادة يجب قراءتها إذا كنت فعلاً تريد أن تستفيد منه. لقد وضعت عشرات بل مئات الساعات في إعداد هذا الكتاب لإظهاره بالمظهر والمضمون اللائق. وتأكد بان كل موضوع تراه بل كل صفحة فيه سبق وقرأتها عشرات المرات لأتأكد أنها بالمستوى المرجو. حظاً موفقاً ، وأرجو أن ينال هذا الكتاب من الغلاف للغلاف حظه في الإعجاب لديكم.

عبدالناصر الكعبي.

الجزء الأول

الأساسيات

بسم الله الرحمن الرحيم

الأساسيات

قد تكون الكلمات الأولى لأي كتاب هي أهم الكلمات وذلك لأن القارئ يبدأ بها فإن أعجب بمضمونها اكمل القراءة وإلا فإن الكاتب قد فشل في توصيل الفكرة المطلوبة بشكل ناجح. لذا أرجو من الله أن يوفقني في جذب انتباهك عزيزي القارئ لكل الكتاب وليس فقط للكلمات الأولى. وهذه النوعية من الكتب ، والتي يطلق عليها الكتب الفنية، يكون من السهل فيها فقد انتباه القارئ. ذلك لأنها تحتوي على الكثير من المعلومات التي قد تبدوا أحيانا اكبر من قدرة استيعاب القارئ في وقت واحد ، فيبدأ الشعور بالملل وفقد الانتباه. فإن حصل لك عزيزي القارئ ذلك فإن أفضل ما تقوم به هو التوقف عن القراءة ، ثم محاولة استيعاب المواضيع التي سبق ومررت بها. أو اذهب للجري قليلا أو اشرب كوبا من الشاي أو اقفز في البحر أو..... والفكرة ببساطة هي أن تأخذ قسطا من الراحة ثم تعود لتكمل ما بدأت، وبهذه الطريقة فإن الإحساس بالملل سوف يختفي تدريجياً، ولكن كن حذرا من أن تتوقف لمدة طويلة لأنه من السهل أن تنسى المواضيع التي مررت بها فيما لو توقفت مدة طويلة عن القراءة.

في هذا الكتاب سنتناول الكثير من المواضيع المختلفة مثل إنشاء الحركة على الكمبيوتر برمجة الأصوات ، برمجة الأبعاد الثنائية ، برمجة الأبعاد الثلاثية، استخدام أدوات الإدخال مثل لوحة المفاتيح ، الفأرة وعصا الألعاب. سوف نتعرف على تكنولوجيا DirectX من شركة مايكروسوفت وكيفية برمجتها والاستفادة منها. الجيد في الأمر أن كل هذه المواضيع مرتبطة ببعضها البعض ، ومعرفة بعضها سوف

يساعدنا على فهم البعض الآخر. ولكي يمكننا استغلال قوة DirectX فإننا سوف نحتاج إلى معرفة ما يلي :

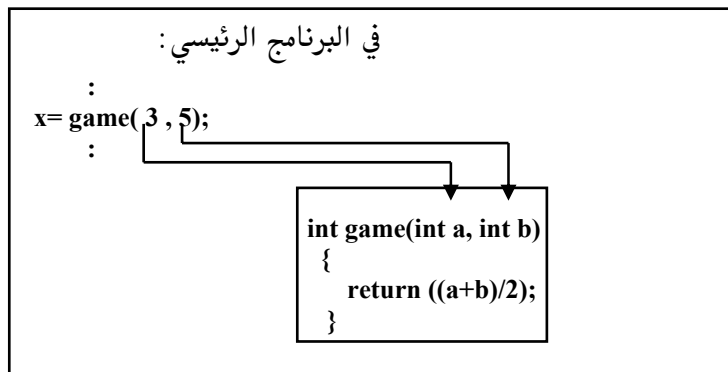
- لغة السي المحسنة (C++) وبما أنها ليست موضوع هذا الكتاب سوف نفترض معرفة القارئ لها (طبعاً ليس المقصود هنا احتراف اللغة وإنما معرفة الصفات الأساسية لها) وسوف نحاول أن نتناول الأساسيات في هذا الفصل.
- برمجة Win32 وهي نظام البرمجة المستخدم من قبل النظام windows وسوف نقوم بشرح تفصيلي لكيفية إنشاء برنامج متكامل باستخدامها في الفصل الثالث.
- بالنسبة للمهتمين بكيفية عمل تكنولوجيا DirectX فإننا في الجزء الثاني من هذا الكتاب سنقوم بشرح تفصيلي لهذه التكنولوجيا (سوف نفصل عمل العنصر DirectDraw وعمل العنصر Direct3D) وسنستخدم البرمجة الكائنية (المقصود بالبرمجة الكائنية هي استخدام الكائنات "Class Objects" في برامجنا).

ملاحظة : ليس المقصود من هذا الفصل تعليمك لغة السي المحسنة ++ C ولكن فقط تذكير بالأساسيات. ولمعرفة أكثر بلغة السي الرجاء مراجعة أحد الكتب الكثيرة والمختصة بتعليمها.

وتذكر أنه في لغة السي المحسنة (كما في بقية لغات الكمبيوتر) فإن كل سطر يعبر عن أمر معين , ويجب أن ينتهي بالإشارة " ; " وإلا فإن المصرف لا ينظر له على أنه انتهى حتى يرى هذه الإشارة. وبالتالي يقوم الكمبيوتر بالنظر للسطر الذي يليه حتى يرى تلك الإشارة وإلا الذي يليه وهكذا حتى يجدها وعندما يجدها فإنه سوف يحسب كل تلك الأسطر السابقة على أنها سطر واحد فقط .

الوظائف Functions

توجد الوظائف في كل لغة من لغات الكمبيوتر كأحد العناصر المهمة، مثل `GOSUB` ، `procedures` ، `subroutines` و `Functions` وجميعها تقوم بنفس العمل وهو نقل التحكم بشكل مؤقت إلى جزء من البرنامج ثم العودة إلى نفس المكان. في لغة السي المحسنة C++ هناك فقط عنصر واحد يقوم بهذه العملية وهو "الوظيفة" أو "Function". وجميع الوظائف تُرجع قيمة معينة إذا لم تكن من نوع `void` (نقوم بإضافة الكلمة `void` قبل اسم الوظيفة عند إنشائها ("function name" `void`)) لتكون من هذا النوع الذي لا يقوم بإرجاع أي قيمة). حتى لو كانت الوظيفة ترجع قيمة معينة فإن المبرمج له حرية استعمال تلك القيمة أو إهمالها. من المؤكد أنك استخدمت الوظائف إذا سبق لك وبرمجت على أي لغة من لغات الكمبيوتر. وتقوم الوظيفة بأخذ الرقم صفر أو أكبر كأحد أعضائها (هذا يعتمد على طريقة إنشائها) ثم تُرجع قيمة واحدة (نوع القيمة المرجعة أيضا يعتمد على طريقة إنشائها) تستطيع استعمالها في البرنامج الرئيسي. مثلا افرض أننا نريد إنشاء وظيفة تقوم بجمع رقمين ثم تقسيمهما على الرقم (2) فإننا سنقوم بما يلي:



في هذا المثال قمنا باستدعاء الوظيفة `game` (طبعا `game` هو مجرد اسم ويمكن تسمية الوظائف عند إنشائها بأي اسم نريده ، المهم أن يكون معناه له علاقة بها حتى نستطيع فهم عملها) ومررنا الرقمين 3 و 5 كأعضاء لها ، فتقوم الوظيفة بدورها بإرجاع نتيجة جمع الرقمين الصحيحين وتقسيمهما على 2 وتوضع هذه النتيجة في المتغير `x` في برنامجنا الرئيسي. كان بالإمكان طبعا القيام بهذه العملية الحسابية مباشرة في البرنامج الرئيسي (بوضع قيمة $x=(3+5)/2$) ولكن ماذا لو إننا أردنا القيام بعمليات حسابية أكثر تعقيدا واستخدامها عشرات المرات عندها سوف نرى فائدة الوظائف.

ملاحظة : يجب استخدام الأمر `return` في هذه النوعية من الوظائف لإرجاع قيمة معينة إلى البرنامج الرئيسي. ولكن ماذا عن الوظائف التي لا ترجع قيمة (تذكر أن هذه النوعية من الوظائف تتكون من الكلمة `void` قبل اسم الوظيفة) وما هي فائدتها؟ ولكي نقوم بالإجابة على هذا السؤال يجب أن نعرف أنها تنقسم إلى نوعين :

```
#include <studio.h>
void output(void); // هذا هو تعريف الوظيفة
void main() {
    printf("You are inside main\n");
    output();      // هنا قمنا باستدعاء الوظيفة
}

void output(void) // هذه هي الوظيفة
{
    printf("You are inside output function\n");
}
```

❖ النوع الأول هو النوع الذي لا يأخذ قيمة ولا يرجع قيمة ونستخدم الكثير من نوعية هذه الوظائف في برامج هذا الكتاب وفائدتها أنها تقوم بعمل معين ثم تُعيد التحكم للبرنامج الرئيسي. لننظر إلى المثال في المربع في الصفحة السابقة ، في هذا المثال بعد أمر التعريف `#include <studio.h>` (هذا الأمر مهم لاستخدام أوامر الإدخال والإخراج في لغة السي مثل الأمر `printf` في برنامجنا) قمنا أولاً بتعريف الوظيفة `void output(void);` وهذه الخطوة مهمة حتى تكون الوظيفة معروفة لجميع البرنامج (عندما يكبر البرنامج في الحجم فإن من الحكمة وضع جميع التعاريف في ملف خاص بها يحمل الإضافة `.h` ، كما سوف نرى من أمثلة الكتاب فيما بعد).

بعد ذلك قمنا في البرنامج الرئيسي (أي في الوظيفة الرئيسية) `Main()` بطباعة العبارة `"You are inside main"` على الشاشة.

ثم قمنا باستدعاء الوظيفة `output` (لاحظ أننا لم نمرر ولم نستقبل أي قيم) والتي بدورها قامت بطباعة العبارة `"You are inside output function"` على الشاشة.

كذلك نحن استخدمنا الأمر `printf` لإخراج جملة معينة على الشاشة وهذا الأمر هو أمر خاص بلغة السي أما لغة السي المحسنة فإنها تستخدم الأمر `<< cout` ولكن الجميل في لغة السي المحسنة أنها تستخدم جميع أوامر لغة السي ولهذا السبب ستجدنا أحياناً نستخدم أوامر لغة السي والتي تعمل بدون أية صعوبات مع السي المحسنة.

ملاحظة : نستخدم `main()` في بيئة `Dos` ولو أننا في بيئة `windows` فإننا سوف نستخدم `WinMain()` وكلاهما عبارة عن وظيفة من الوظائف تتميز أن الكمبيوتر يبدأ بالنظر إليهما قبل أي وظيفة أخرى ولذلك نسميهما بالوظيفة الرئيسية.

❖ أما النوع الثاني فهو خليط من كلتا الحالتين السابقتين ، فالوظيفة تأخذ قيم ولكن لا ترجع أي قيمة كما في المثال التالي :

```
#include <studio.h>
void output(int a); // هذا هو تعريف الوظيفة
void main() {
    printf("You are inside main\n");
    output(9);        // هنا قمنا باستدعاء الوظيفة
}

void output(int a) // هذه هي بداية الوظيفة
{
    printf("You are inside output function and a= %d\n",a);
}
```

ولاحظ التشابه الكبير بين النوعين فكل ما قمنا به هنا هو تغيير التعريف للوظيفة ليكون " void output(int a);" ثم جعلنا الوظيفة تقوم بطباعة الجملة التالية مع القيمة الممررة على الشاشة " You are inside output function and a=9".
 لاحظ أن تعريف الوظيفة دائما يكون مطابق لبداية الوظيفة مع إضافة الإشارة " ; "

لماذا كل هذا التركيز على الوظائف ؟

لأننا عندما نقوم ببرمجة برنامج ضخم فإن تقسيم البرنامج إلى وظائف متعددة سيساعد على فهمه و يعطينا القدرة على زيادة حجمه بدون قلق (الكثير يفضل استخدام الكائنات والتي بدورها تحتوي على الوظائف كأعضاء لها بدل استخدام الوظائف بشكل مباشر ، ولأن استخدام الكائنات "أو البرمجة الكائنية" تعتبر أحد أهم مميزات لغة السي المحسنة فسوف نطلع على كيفية استخدامها وبرمجتها عن طريق مكتبة AGDX. كما أننا في الجزء الأول من الكتاب سنكتفي باستخدام الوظائف وفيها نستخدم الكائنات بشكل مباشر في أمثلة الكتاب ولأننا بهذه الطريقة نكون اقرب إلى طريقة برمجة النظام DOS مما يتميز بسهولة اكبر سيستطيع الكثير ممن سبق لهم البرمجة على هذا النظام التعرف على برامج الكتاب بشكل سريع وواضح. هناك تعامل كبير مع الكائنات في الجزء الثاني من الكتاب). مثلاً في كرة القدم (أنا لا العب كرة القدم ولكنها تعتبر من اشهر الألعاب الرياضية في العالم لذلك سوف أخذها كمثال) يجب تقسيم الفريق إلى عدة أقسام : قسم دفاع وقسم هجوم و حراسة. كذلك كل قسم له وظيفة معينة مثلاً الهجوم يقسم إلى قسمين أو ثلاثة : هجوم من اليسار ، هجوم من اليمين وهجوم وسط وكذلك الدفاع : دفاع يسار ، دفاع يمين ودفاع وسط. إذاً فبرغم أن لدينا فريق واحد ولكن له أقسام عديدة أو وظائف عديدة وكل وظيفة لها عدة أشخاص (كائنات) معينين يقومون بعمل معين (الهجوم ، الدفاع أو الحراسة في هذه الحالة). كذلك نستخدم نفس القاعدة في برامجنا ، فمع العلم أن البرنامج واحد إلا أن له عدة وظائف وكل وظيفة تستخدم عدة كائنات للقيام بعمل معين وبذلك يسهل التعامل مع البرنامج.

المتغيرات من نوع Struct

عندما نقوم ببرمجة الكمبيوتر فإننا نستخدم الكثير من المتغيرات، في كثيرة من الأحيان تكون هذه المتغيرات ذات علاقة ببعضها البعض. افرض مثلا أننا نريد أن نكتب برنامج لشركة معينة يقوم بأخذ اسم كل عامل وعمره ودخلة الشهري. في هذه الحالة سوف نستخدم ثلاث متغيرات مختلفة (الاسم، العمر، والدخل الشهري) ولكن لاحظ أن هذه المتغيرات الثلاث يربطها شيء واحد مهم وهو أننا نتكلم عن نفس العامل. لذلك نستطيع عن طريق الأمر **struct** أن ننشئ نوع جديد يحتوي على هذه المتغيرات (أو الأعضاء) الثلاث في متغير واحد فقط كما يلي:

```
struct worker
{
char name[10];
int age;
int income;
};
```

الآن اصبح لدينا نوع جديد من المتغيرات سميناه **worker** (طبعا الكلمات **worker** ، **name** ، **age** و **income** هي مجرد أسماء ويمكننا استخدام أي تسمية نشاء) ويحتوي على ثلاث أعضاء من المتغيرات: المتغير العضو **name** من نوع **char** وهو متغير الاسم ثم المتغير العضو **age** من نوع **int** وهو متغير العمر ثم أخيرا المتغير العضو **income** من نوع **int** وهو متغير الدخل الشهري.

الآن بعد أن اصبح لدينا نوع جديد من المتغيرات، كيف يمكننا أن نستخدم هذا النوع الجديد في برنامجنا الرئيسي كل ما علينا فعلة هو إنشاء متغير من هذا النوع الجديد **worker** (طبعا

النوع الجديد "worker" ليس نوع من أنواع المتغيرات المستخدمة في لغة السي المحسنة (مثل int أو float أو short ... الخ) ولكنه مجرد نوع نحن قمنا بإنشائه في الخطوة السابقة)

المتغيرات من نوع enum

الكلمة enum مأخوذة من الكلمة الإنجليزية enumeration وتعني "قائمة" لها ترتيب معين. وكذلك في لغة السي المحسنة فإنها تعني قائمة لعدد من العناصر "المتغيرات" تحمل القيم ابتداء من الصفر : 0, 1, 2, 3, إلى قيمة ترتيب آخر عنصر بها. أفرض مثلا انك تريد استخدام أيام الأسبوع في برنامجك (الجمعة ، السبت ، الأحد ، الاثنين ، الثلاثاء ، الأربعاء ، والخميس) لن يكون من الحكمة استخدام المتغيرات من نوع char للتعبير عن كل اسم لكل يوم في البرنامج ، وسيكون من الأفضل إعطاء رقم للتعبير عن كل يوم وبذلك نستطيع التعامل مع الأرقام بدل استخدام المتغيرات من نوع char. واستخدام نوعية enum يجعل من هذه الطريقة اكثر سلاسة بأنه لا يعطي فقط رقما لكل يوم بل أنه كذلك يسمح لنا باستخدام الاسم لليوم بدل استخدام الرقم :

```
enum {
Friday,
Saturday,
Sunday,
Monday,
Tuesday,
Wednesday,
Thursday
};
```

الآن في برنامجنا نستطيع استخدام أسماء أيام الأسبوع للتعبير عنها.

الذاكرة

يحتوي الكمبيوتر الشخصي على ثلاث أنواع رئيسية من الذاكرة. النوع الأول وهي ذاكرة المعالج الرئيسي وتعرف بذاكرة الكاش ، وتكون موجودة كجزء من المعالج الرئيسي وهي خارج مجال اهتمامنا. والنوع الثاني وهي ذاكرة النظام ، وتكون موجودة على اللوحة الأم (motherboard). وتحتوي الكمبيوترات الشخصية اليوم على ما لا يقل عن 32 ميجابايت (على الأقل في الكمبيوترات التي تعمل على نظام windows95/98 أو windows2000). والنوع الثالث هي ذاكرة الفيديو (أحيانا أطلق عليها ذاكرة العرض) وتكون موجودة على بطاقة الفيديو ويعتبر 2 ميجابايت من هذه الذاكرة هو المعدل الأدنى اليوم في الكمبيوترات الشخصية. لماذا الآن نتكلم عن الذاكرة ؟ خلال برمجة الألعاب والصوت والصورة سوف تلاحظ أننا نقوم باستخدام الذاكرة في تخزين المعلومات المطلوبة خلال عمل البرنامج. لذلك يجب أن تكون لدينا فكرة عن كيفية التحكم فيها لكي نستطيع استخدامها بالشكل الصحيح. سوف نكتشف كذلك أن العنصر DirectDraw من عناصر DirectX هو في الحقيقة المُدير لذاكرة الفيديو وسوف نستخدمه للتحكم في ذاكرة العرض (أو الفيديو).

في لغة السي المحسنة نقوم باستخدام الأمر "new" لحجز الذاكرة المطلوبة للرسوم والصور المستخدمة في وقت واحد في البرنامج. ولكن استخدام هذا الأمر بدون انتباه قد يؤدي إلى حجز قدر من الذاكرة أكبر من الحد الأدنى وهو 2 ميجابايت. وفي حالة القيام بتشغيل البرنامج على كمبيوتر يحتوي فقط على هذا الحد الأدنى ، فإن العنصر "DirectDraw" يقوم باستخدام ذاكرة النظام ليتفادى مشكلة الذاكرة المحدودة. وهذا جميل ولكن ينتج عنه مشكلة وهي "السرعة". نعم السرعة ، فنقل

المعلومات من ذاكرة النظام الموجودة على اللوحة الأم إلى بطاقة الفيديو يمر عن طريق ممر يسمى باص "Bus" له سرعة معينة. وهذه عملية بطيئة مقارنة لسرعة نقل المعلومات من ذاكرة الفيديو الموجودة على نفس البطاقة. ولهذا السبب وجدت ذاكرة الفيديو في الأصل. فكما عرفنا أن ذاكرة العرض تقع في بطاقة الفيديو وهي البطاقة المهتمة بكل ما يعرض على الشاشة من صور وحركة، لذلك يوجد على هذه البطاقة معالج مسؤول عن العرض. ووجود ذاكرة العرض على نفس البطاقة وليس على اللوحة الأم للكمبيوتر يوفر الكثير من الوقت للمعالج في إيجاد المعلومات المطلوبة. وهذا ينتج عنه زيادة كبيرة في سرعة العرض. ولذا عندما يحتاج معالج العرض للبحث عن المعلومات في ذاكرة النظام فإن ذلك يأخذ وقت أكبر وبالتالي سرعة أقل عما لو كانت تلك المعلومات على ذاكرة العرض. لهذا السبب يجب استخدام الأمر "new" بكثير من الحذر.

في الواقع أغلب الكمبيوترات التي تباع اليوم تحتوي على 4 ميجابايت أو أكثر من ذاكرة العرض وقد تسأل عزيزي القارئ لماذا كل هذا الاهتمام لذاكرة العرض بالذات؟ ولماذا نحبذ أن تكون بأكبر قدر ممكن؟ ولإجابة هذا السؤال نحن نعلم بأن برامج الملتيميديا (Multimedia) والألعاب تستخدم الصور على الشاشة لتقوم بعملها. ونحن نعلم كذلك بأن تعبئة تلك الصور في ذاكرة الكمبيوتر يأخذ جزء معين من الوقت. لذلك نقوم بتعبئة الصور المطلوب استخدامها في وقت واحد في ذاكرة الفيديو، وهكذا لا يوجد داعي للانتظار لتعبئة تلك الصور كل ما احتجنا إليها عند عمل البرنامج. لأنها أودعت سابقا في ذاكرة العرض. وكلما زاد حجم تلك الذاكرة كلما كان بالإمكان استخدام صور أكثر وبالتالي استخدام مؤثرات أكثر على الشاشة.

ملاحظة: عند استخدام الأمر "new" في لغة السي المحسنة فإن الكمبيوتر يقوم بحجز الذاكرة المطلوبة، ويجب استخدام الأمر "delete" لتحرير تلك الذاكرة عند الانتهاء منها. وإلا فإنها سوف تبقى في ذاكرة الكمبيوتر حتى نطفي الجهاز وهذا يسبب الكثير من المشاكل. (سننظر إلى هذين الأمرين بشكل أدق فيما بعد)

لننظر إلى هذا المثال :

```

struct data
{
    int month;
    int day;
    int year;
};

int main()
{
    struct date *my_date;
    my_date=new date;
    my_date->month =8;
    my_date->day=22;
    my_date->year=1956;
    :
    delete(my_date);
}
    
```

لنفرض أننا نريد أن نقوم ببرنامج يحفظ تواريخ معينة في الذاكرة بحيث يحتوي كل تاريخ على اليوم والشهر والسنة، أولاً قمنا بإنشاء متغير من نوع "Structure" وسميناه date وجعلناه يحتوي على ثلاث عناصر ، month ، day ، و year بعد ذلك في الأمر "main()" قمنا بإنشاء مؤشر إلى المتغير "date" ثم قمنا بحجز جزء من الذاكرة لذلك المتغير (عن طريق الأمر "new") بعدما قمنا بذلك اصبح المتغير في

الذاكرة و أصبحنا نستطيع أن نستخدم عناصر ذلك المتغير عن طريق المؤشر الذي أنشأناه (مثلا قمنا بوضع الرقم 8 في العنصر month والرقم 22 في العنصر day والرقم 1956 في العنصر year وهكذا) الآن فإن المتغير "my_date" يحتوي على المعلومات 22-8-1956. وعند الانتهاء من استخدام الذاكرة نقوم بتحريرها عن طريق استخدام الأمر "delete"

عندما نستخدم الأمر "new" كل ما علينا فعله هو إعطاء نوع المتغير (استخدمنا date في مثالنا هذا) ويعرف الكمبيوتر بعد ذلك الحجم المطلوب بالبايت.

- في البرنامج السابق هل من الممكن القيام بنفس العملية بدون استخدام الأمرين New و delete فإذا كانت الإجابة نعم، إذا لماذا نستخدمها؟
- إجابة القسم الأول من السؤال هو "نعم" يمكن كتابة البرنامج السابق للقيام بنفس العملية بدون استخدام الأمرين New و delete كما يلي :

```
struct data
{
int month;
int day;
int year;
};

int main()
{
struct date my_date;
my_date.month=8;
my_date.day=22;
my_date.year=1956;
:
}
```

لاحظ أننا في هذه الحالة عندما نريد أن نستخدم أحد عناصر المتغير date نستخدم النقطة "." بدل السهم ">" في الحالة السابقة (وهذه القاعدة تنطبق على الكائنات

كذلك لأن الكائنات عبارة عن شكل مطور للمتغيرات من نوع "Structure" كما سوف نرى فيما بعد). الآن نأتي إلى القسم الثاني من السؤال "إذا لماذا نستخدمها؟" صحيح أن البرنامج الثاني يقوم بنفس عمل البرنامج الأول ولكن لو سألنا أنفسنا أين يقوم البرنامج الثاني بتخزين المعلومات؟ والإجابة هي الذاكرة، إذا كلا البرنامجين يقومان بتخزين المعلومات في الذاكرة. الفرق الوحيد أننا في البرنامج الأول نحن الذين أمرنا الكمبيوتر بتخزين المعلومات في الذاكرة أما في البرنامج الثاني فإن الكمبيوتر هو الذي قام بتلك العملية من تلقاء نفسه. ولذلك في البرنامج الأول فإن الكمبيوتر لن يقوم في أي حالة بتحرير الذاكرة إلا عندما نأمره بذلك أما في البرنامج الثاني فإن الكمبيوتر يقوم بتحرير الذاكرة متى أراد. هنا تتضح المشكلة ، ففي حالة لو أن الكمبيوتر يقوم بحجز الذاكرة وتحريرها كما يشاء نفقد نحن كمبرمجين قدرتنا على التحكم في سرعة الكمبيوتر (تذكر أن تعبئة المعلومات في الذاكرة يأخذ جزء من الوقت) ولذلك نفضل نحن كمبرمجين أن نقوم باستخدام البرنامج الأول لأمر الكمبيوتر بحجز الذاكرة وتحريرها حسب مشيئتنا.

طبعاً البرنامجين السابقين لا يشكلان أي مشكلة في السرعة لأن كلاهما سهل التركيب ولكن عند إنشاء برنامج ضخم يقوم باستخدام جميع ذاكرة العرض كما سترى في الأمثلة فيما بعد ، فالفرق سيكون واضح.

أنت تعرف الآن ماذا تعني كلمة "Loading" أو تعبئة عند بداية كل لعبة للكمبيوتر أو لعبة فيديو، فإن ما يقوم به البرنامج هو كما سبق وشرحنا تعبئة المعلومات المطلوب استخدامها في وقت واحد على ذاكرة العرض أو ذاكرة النظام.

بطاقات AGP

في الآونة الأخيرة قامت الشركات بإدخال تكنولوجيا جديدة لبطاقات الفيديو تسمى AGP (مختصر الجملة : Accelerated Graphics Port) هي عبارة عن باص أو موصل بين بطاقة الفيديو واللوح الأم (شبيهه بالتكنولوجيا القديمة ISA و EISA و PCI) وتقوم هذه التكنولوجيا بنقل المعلومات بين الذاكرة الموجودة على اللوحة الأمر "ذاكرة النظام" وبين بطاقة العرض بشكل سريع جدا (100MHz) يصل إلى سرعة الذاكرة الموجودة على بطاقة الفيديو "ذاكرة الفيديو" بحيث أصبحت الحاجة إلى النوعية الأخيرة من الذاكرة غير مهمة حتى إن بعض البطاقات الجديدة التي تستغل هذه التكنولوجيا لا تحمل أي ذاكرة فيديو عليها. صممت هذه التكنولوجيا أساسا لحل مشكلة السرعة في البرامج ثلاثية الأبعاد والتي تستخدم العنصر DIRECT3D من عناصر DirectX ولأن هذا العنصر يقوم أساسا باستعمال العنصر DirectDraw للتعامل مع الذاكرة والقيام بعملية فإن كلاهما سوف يستفيد من هذه التكنولوجيا الجديدة. ولسوء الحظ أن هذه التكنولوجيا لازالت في بدايتها وليست متوفرة في اغلب الكمبيوترات. والاعتماد عليها في برمجتنا يكون نوع من المخاطرة، كذلك نحن لا نعرف المدة التي سيأخذها منتجين بطاقات الفيديو للقيام باستغلال هذه التكنولوجيا. وعليه لن نقوم باستخدامها في كتابنا هذا ولكن هناك شرح في الجزء الثاني من هذا الكتاب في كيفية استغلالها لو أردنا (في الجزء المختص بإنشاء العنصر DirectDraw).

حفظ المعلومات على القرص الصلب

بعد أن تعرفنا على كيفية التعامل وتخزين المعلومات في ذاكرة الكمبيوتر كثيرا ما نرى الحاجة إلى تخزين المعلومات لبرنامج معين على القرص الصلب ثم استعادتها مرة أخرى إلى البرنامج ، خاصة في برامج الصوت والصورة التي نحن بصدد التعرف عليها في هذا الكتاب. أفرض مثلاً أننا قمنا ببرمجة لعبة وأراد اللاعب أن يتوقف في منتصف اللعبة ثم يكمل فيما بعد ، في هذه الحالة نرى الحاجة إلى تخزين بعض المعلومات في ملف خاص حتى إذا بدأ اللاعب البرنامج في وقت آخر يستطيع أن يبدأ من النقطة التي توقف عندها.

أن أفضل طريقة لتخزين العديد من المتغيرات لبرنامج معين في ملف على القرص الصلب ولاستعادتها لاحقاً هو وضعها في كائن واحد. ولتسهيل العملية سنستخدم المتغيرات من نوع `struct` (نحن نعلم كذلك أن هذه المتغير هو عبارة عن صورة مبسطة للكائنات) لأننا نستطيع عن طريقها أن نضع مجموعة من المتغيرات المختلفة وبقيم مختلفة في متغير واحد بحيث أن كل ما علينا فعله هو قراءة هذا المتغير من الملف عند الحاجة إليه.

ملاحظة : الطريقة التي نشرحها هنا ليست الطريقة الوحيدة في بيئة `windows` لتخزين المعلومات على القرص الصلب ، ولكنها في رأيي أفضل طريقة بدون الحاجة إلى استخدام مكتبات `MFC`.

مثال (هذا المثال موجود على الملف SAVE1 على القرص المدمج):
 أولاً ليكون المثال مفيد دعونا نفترض أننا في برنامجنا الرئيسي نملك هذا المتغير من نوع struct وسوف نطلق عليه الاسم "GAME_VARS" كما يلي:

```
struct GAME_VARS
{
    int nPlayerPositionX; ← موقع لاعب على المحور الصادي
    int nPlayerPositionY; ← موقع لاعب على المحور السيني
    int nScore; ← النتيجة
    int nLevel; ← المرحلة
    int array[10]; ← متغير من أعداد صحيحة
    char szMapTiles[4000]; ← متغير من نوع أحرف
};
```

طبعاً أنت تستطيع استخدام المتغيرات التي تراها مناسبة لبرنامجك وهذا مجرد مثال نحاول أن نستخدم فيه أكبر عدد ممكن من المتغيرات المختلفة لترى كيف تستطيع أن تحفظها على القرص الصلب وكيف تستعيدها مرة أخرى.
 حسناً ، الآن نريد أن نحفظ هذا المتغير من نوع Struct بكل محتوياته على القرص الصلب ، وسوف نستخدم عدة أوامر للقيام بذلك. كما أننا سوف نقوم بكتابة وظيفتين : الأولى لحفظ المعلومات "SaveGame()" والأخرى لاسترجاعها مرة أخرى من القرص الصلب "LoadGame()".

الوظيفة SaveGame() :

أولا دعونا نرى كيفية تخزين المعلومات ، وسوف نقوم بإنشاء هذه الوظيفة لتقوم بهذه العملية بحيث نستطيع استخدامها في برامجنا فيما بعد.

سنقوم أولا بإنشاء متغير آخر من نوع struct وفيه نقوم بإنشاء متغير من النوع "GAME_VARS" الذي أنشأناه منذ لحظات نسميه المتغير "gv" ويحتوي على جميع المتغيرات التي سبق وأنشأناها في "GAME_VARS" كما يلي :

```
struct SAVE_GAME_FILE
{
    GAME_VARS gv;
};
```

بمعنى أخرى هذا المتغير الجديد والذي سميناه "SAVE_GAME_FILE" ما هو إلا نسخة طبق الأصل عن المتغير "GAME_VARS" (سوف يختلف فيما بعد عندما نقوم ببعض الإضافات إليه "عندما نقوم بإضافة وظيفة () GetChecksum لحماية الملف")

الآن دعونا ننشئ متغير آخر من نوع "GAME_VARS" سوف نطلق عليه "gv" (هذا متغير مختلف تماما عن المتغير السابق والذي يحمل نفس الاسم لأن ذلك المتغير كان جزء من "SAVE_GAME_FILE") كما يلي :

```
GAME_VARS gv;
```

بعد ذلك نقوم بإنشاء وظيفة الحفظ "SaveGame()" ونجعلها تُرجع القيمة "صحيح" أو القيمة "غير صحيح" في حالة حصول أي خطأ. وهذه الوظيفة تمرر اسم الملف المراد تخزينه (pszFilepath):

```
bool SaveGame(char *pszFilepath)
{
    SAVE_GAME_FILE sgf; ← A
    FILE *hfile; ← B
    size_t tWritten; ← C

    sgf.gv=gv; ← D

    hfile=fopen(pszFilepath, "w"); ← E
    if (!hfile) return false; ← F
    tWritten=fwrite( (void *) &sgf, sizeof(sgf), 1, hfile); ← G
    if (0!=fclose(hfile)) return false; ← H
    return (1==tWritten); ← I
}
```

الشرح حسب ترتيب الأحرف الإنكليزية :

A. قبل الدخول في شرح عمل هذه الوظيفة نحن نعلم بأننا الآن نملك متغيرين من نوع struct (SAVE_GAME_FILE و GAME_VARS) وإستخدمنا الأول في إنشاء المتغير العام "gv" وسوف نستخدم الثاني في إنشاء المتغير "sgf" (الآن كلا المتغيرين "gv" و "sgf" يحتويان على نفس المتغيرات) كما في الخطوة "A".

B. بعد ذلك نقوم بإنشاء متغير "hfile" ليكون مؤشر الملف الذي سوف نقوم باستخدامه للتخزين، كما في الخطوة "B".

C. ثم نقوم بإنشاء متغير "tWritten" من نوع size_t نستخدمه لتخزين القيمة المرجعة عند كتابة المعلومات للملف (فيما بعد عن طريق الأمر (fwrite) كما في الخطوة "C".

D. في الخطوة "D" نقوم بوضع قيم محتويات العضو "sgf.gv" لتساوي قيم محتويات المتغير "gv" حتى نتأكد من تطابقيهما.

E. في الخطوة "E" نقوم بفتح ملف للكتابة وإعطاءه الاسم الذي تأخذه وظيفة التخزين عن طريق الأمر "fopen(pszFilepath, "w")" و الحرف "w" يعني أن الملف سوف يفتح للكتابة فقط. ويرجع هذا الأمر عنوان الملف في الذاكرة فنقوم بوضعه في مؤشر الملف "hfile".

F. في الخطوة "F" نقوم بالتأكد (عن طريق التأكد بأن قيمة المؤشر "hfile" لا تساوي صفر) أن عملية فتح الملف قد تمت بنجاح.

G. في الخطوة "G" بعد أن فتحنا الملف للكتابة نقوم بعملية الكتابة عن طريق استخدام الأمر "fwrite()" وتمرير مؤشر الملف وحجمه.

H. في الخطوة "H" نقوم بإغلاق الملف والتأكد بأن عملية الإغلاق مرت بنجاح وإلا فنرجع "غير صحيح"

I. في الخطوة الأخيرة "I" نرجع القيمة صحيح.

في الحقيقة هذه الوظيفة بسيطة التركيب ولكنها فعالة في حفظ المعلومات المطلوبة على القرص الصلب. وتكمن فعاليتها باستخدام الأمر "fwrite()" لتقوم بعملها.

الوظيفة LoadGame() :

بعد أن تعرفنا على وظيفة تخزين المعلومات في ملف على القرص الصلب ، دعونا الآن نرى كيف نستطيع أن نجلب تلك المعلومات مرة أخرى لبرنامجنا وكيف يمكننا استخدامها. ولجعل الأمور في مسارها الصحيح سوف نقوم بإنشاء وظيفة تعبئة تقوم بهذه العملية وهي الوظيفة LoadGame() كما يلي :

```
bool LoadGame(char *pszFilepath)
{
    SAVE_GAME_FILE sgf;
    FILE *hfile;
    size_t tRead;

    hfile=fopen(pszFilepath, "r"); ← A
    if (!hfile) return false;
    tRead=fread( (void *) &sgf, sizeof(sgf), 1, hfile); ← B
    fclose(hfile);
    if (tRead!=1) return false;

    gv=sgf.gv; ← C

    return true;
}
```

أنظر إلى التشابه الكبير بين وظيفة التخزين وهذه الوظيفة ، فنحن هنا قمنا تقريبا بعكس العملية كما يلي :

- بدلاً من فتح ملف للكتابة كما فعلنا في وظيفة التخزين قمنا بفتح ملف للقراءة كما في الخطوة "A" حيث استخدمنا الأمر "fopen(pszFilepath, "r")" و الحرف "r" يعني أن الملف سوف يفتح للقراءة فقط.

- استخدمنا في هذه الوظيفة أمر القراءة fread() (يقوم بوضع جميع القيم التي سبق وخبزناها في المتغير sgf) في الخطوة "B" بدل أمر الكتابة "fwrite()" في وظيفة التخزين السابقة.
 - وأخيراً في الخطوة "C" قمنا بوضع القيم في المتغير "gv" بحيث نستطيع الآن استخدامه في برنامجنا (جميع القيم الموجودة في الملف ستكون موجودة في هذا المتغير).
- بذلك نرى أن وظيفة التعبئة هذه تقوم بعكس وظيفة التخزين واستخدام كلا الوظيفتين سهل للغاية كما سنرى.

مثال على كيفية استخدام وظيفة التخزين ووظيفة التعبئة:

```
extern GAME_VARS gv; ← A

void main()
{
gv.nPlayerPositionX=245; ← B
gv.nPlayerPositionY=333; ← C

SaveGame("newsave.ttt"); ← D

gv.nPlayerPositionX=0; ← E
gv.nPlayerPositionY=0; ← F

LoadGame("newsave.ttt"); ← G

printf( "%d\n",gv.nPlayerPositionX ); ← H
printf( "%d\n",gv.nPlayerPositionY ); ← I
}
```

في هذا المثال سوف نرى كيف نستطيع استخدام الوظيفتين السابقتين في حفظ المعلومات المطلوبة في ملف وسوف نقوم بتسميته "newsave.ttt" (طبعاً اسم الملف

متروك لك ، وهذا الاسم هو مجرد مثال) ثم نقوم بتعبئة المعلومات مرة أخرى للبرنامج ونرى هذه العملية كما يلي :

- في الخطوة "A" نقوم بتعريف المتغير "gv" (والذي سبق وعرفناه فيما قبل ، ووضعنا فيه المتغيرات التي نريد تخزينها ، طبعا يمكنك وضع المتغيرات المناسبة لبرنامجك).

- في الخطوتين "B" و "C" قمنا بإعطاء بعض أعضاء المتغير "gv" قيم معينة.

- في الخطوة "D" قمنا بتخزين المتغير "gv" عن طريق الأمر SaveGame() (أصبحنا نطلق عليه "أمر" وليس "وظيفة" لأن أغلب أوامر لغة السي المحسنة هي وظائف سبق وبرمجت لنا كما هو الحال هنا مع وظيفة التخزين).

- في الخطوتين "E" و "F" قمنا بإعطاء نفس أعضاء المتغير "gv" والتي قمنا بتخزينها قيمة الصفر (حتى نستطيع اختبار أمر التعبئة LoadGame() لإعادة القيم التي سبق وحزناها في الملف "newsave.ttt" عن طريق أمر التخزين ("SaveGame()")

- في الخطوة "G" وبعد أن تأكدنا من تغيير القيم السابقة لأعضاء المتغير "gv" قمنا بتعبئة القيم القديمة من الملف "newsave.ttt"

- في الخطوتين "H" و "I" قمنا بإظهار القيم على الشاشة حتى نتأكد أنها فعلا القيم التي سبق وحزناها وذلك عن طريق أمر الإظهار printf().

الخلاصة:

عزيزي القارئ خلاصة هذا الموضوع أننا باستخدام الوظيفتين "SaveGame()" و "LoadGame()" في برنامجنا هو أن نقوم بحفظ وتعبئة المتغير "gv" (والذي نستطيع أن نضع فيه ما شئنا من المتغيرات المطلوب حفظها) من وإلى القرص الصلب.

حفظ المعلومات من التغيير :

(موجود على الملف SAVE2 على القرص المدمج)

ما سبق وشرحناه هو كل ما نحتاجه لتخزين المعلومات على القرص الصلب واستخدامها فيما بعد ، ولكن في الواقع الأمور ليست كما نريد ، فنحن المبرمجين نقوم ببرمجة برامجنا للآخرين لاستعمالها. وبما أننا نقوم بحفظ المعلومات في ملف على القرص الصلب فإنها معرضة للتغيير. أليست كل الملفات معرضة للتغيير ؟ نعم ، هذا صحيح ولكن أي ملف إذ تم تغييره فإنه ببساطة لا يعمل ولكن ملفنا هنا قابل للتغيير وللأسف فإنه من الممكن إعطاء المتغيرات المحفوظة قيم ليست متوقعة. مثلاً لو أننا صممنا لعبة معينة، وفي هذه اللعبة هناك مرحلتين نعبر عنهما بمتغير يحمل القيمة "1" أي المرحلة الأولى أو القيمة "2" أي المرحلة الثانية. فعند تعبئة المستعمل للملف المحفوظ فإن قيمة المتغير ستكون إما "1" أو "2" وأي قيمة أخرى سوف تنتج عنها مشاكل عدة. لذلك نحن نحتاج إلى حفظ ملفاتنا من التغيير ولهذا سوف نقوم بإنشاء عملية تقوم بحفظ حجم الملف المحفوظ سابقاً وعند تعبئته تقوم بالتأكد أن الملف لم يتغير حجمه "بمعنى أنه لم يتم تغييره بطريقة أو بأخرى من قبل الآخرين". وكذلك سوف نحتاج إلى بعض التغييرات (أو بالأحرى الإضافات) في كلا والوظيفتين (الحفظ والتعبئة) حتى نستفيد من هذه الوظيفة الإضافية الجديدة عند حفظ وتعبئة الملف.

الوظيفة () GetChecksum :

```

long GetChecksum( BYTE *pv, long lSize)
{
    long lCount=0;
    BYTE *pSeek=pv, *pLast=pv+lSize;

    while (pSeek!=pLast) {lCount+=(int) *(pSeek++);}

    return lCount;
}

```

تقوم هذه الوظيفة بقياس حجم الملف بالبايت ثم إرجاعه. وبالتالي نستطيع استخدامها لحفظ حجم الملف في عضو من الأعضاء عند حفظ الملف، ثم نقوم بمقارنة قيمة ذلك العضو بحجم الملف عند محاولة تعيئته كما نرى هنا :

سنقوم أولاً بإضافة عضوين جديدين للمتغير "SAVE_GAME_FILE" والذي سبق وشرحنه أنه نسخة طبق الأصل عن المتغير "GAME_VARS" وأشرنا أنه سوف يختلف فيما بعد عندما نقوم بإضافة وظيفة () GetChecksum لحماية الملف وبما أننا الآن نقوم بالتعامل مع هذه الوظيفة الجديدة وسوف نقوم بإضافة العضو "lChecksum" و العضو "szFileID[9]" وذلك حتى نستطيع أن نحفظ حجم الملف

به، ونرى المتغير كما يلي :

```

struct SAVE_GAME_FILE
{
    char szFileID[9];
    long lChecksum;
    GAME_VARS gv;
};
#define SAVE_GAME_FILE_ID "SAVEFILE"

```

ثم باستخدام الأمر #define وضعنا الجملة "SAVE_GAME_FILE_ID" تساوي الجملة "SAVEFILE" وليس هناك سر في ذلك ونما فعلنا هذا لقصد التوضيح فقط.


الوظيفة SaveGame() بعد استغلال الوظيفة الجديدة () GetChecksum :

```
bool SaveGame(char *pszFilepath)
{
    SAVE_GAME_FILE sgf;
    FILE *hfile;
    size_t tWritten;

    strcpy(sgf.szFileID, SAVE_GAME_FILE_ID);
    sgf.lChecksum=GetChecksum((BYTE *) &gv, sizeof(gv));

    sgf.gv=gv;

    hfile=fopen(pszFilepath, "w");
    if (!hfile) return false;
    tWritten=fwrite( (void *) &sgf, sizeof(sgf), 1, hfile);
    if (0!=fclose(hfile)) return false;
    return (1==tWritten);
}
```



كما نرى هنا أن الإضافات هي كما يلي :

```
strcpy(sgf.szFileID, SAVE_GAME_FILE_ID);
```

أولاً قمنا باستخدام الأمر strcpy ليقوم بنسخ محتويات SAVE_GAME_FILE_ID وتحتوي كما اطلعنا عليها سابقاً على الجملة "SAVEFILE" إلى المتغير szFileID أحد أعضاء sgf.

بعد ذلك حصلنا على حجم الملف عن طريق الوظيفة GetChecksum() ووضعناه في المتغير IChecksum أحد أعضاء sfg كما يلي :

```
sgf.IChecksum=GetChecksum((BYTE *) &gv, sizeof(gv));
```

هذه هي كل الإضافات في هذه الوظيفة.

الوظيفة LoadGame () بعد استغلال الوظيفة الجديدة GetChecksum () :

```
bool LoadGame(char *pszFilepath)
{
    SAVE_GAME_FILE sfg;
    FILE *hfile;
    size_t tRead;

    hfile=fopen(pszFilepath, "r");
    if (!hfile) return false;
    tRead=fread( (void *) &sfg, sizeof(sfg), 1, hfile);
    fclose(hfile);
    if (tRead!=1) return false;

    if (strcmp(sfg.szFileID, SAVE_GAME_FILE_ID)!=0) return false;
    IChecksum=GetChecksum((BYTE *) &(sfg.gv), sizeof(sfg.gv));
    if (IChecksum!=sfg.IChecksum) return false;

    gv=sfg.gv;

    return true;
}
```

الآن نحن جاهزون لتعبئة الملف مرة أخرى ولكن هذه المرة بأخذ الوظيفة GetChecksum () الجديدة في الاعتبار. وكما ترى أن الإضافات هنا هي كما يلي :

```
if (strcmp(sfg.szFileID, SAVE_GAME_FILE_ID)!=0) return false;
```


أولاً قمنا باستخدام الأمر `strcmp` ليقوم بمقارنة محتويات المتغير `szFileID` أحد أعضاء `sgf` مع `SAVE_GAME_FILE_ID` والذي يجب أن يحتوي كما اطلعنا عليه سابقاً على الجملة "SAVEFILE". إذا كان كل شيء على ما يرام نذهب للأمر الذي يليه وإلا فإننا نرجع القيمة "غير صحيح".

بعد ذلك نقوم بالتأكد على أن حجم الملف المحفوظ يساوي قيمة المتغير `IChecksum` أحد أعضاء `sgf`. عن طريق استخدام الوظيفة `(GetChecksum)` للحصول على حجم الملف الذي نحن بصدده قراءته. ونرى ذلك كما يلي :

```
IChecksum=GetChecksum((BYTE *) &(sgf.gv), sizeof(sgf.gv));
if (IChecksum!=sgf.IChecksum) return false;
```

إذا كان كل شيء على ما يرام نذهب للأمر التالي وإلا فإننا نرجع غير صحيح. هذه هي كل الإضافات في هذه الوظيفة.

تعطيل عمل حافظ الشاشة

بما أننا نتعامل مع نظام `windows` فإن هذا النظام يقوم باستخدام برنامج حافظ للشاشة (`screen saver`) يقوم بالعمل إذا لم يتم المستعمل بالضغط على مفتاح من لوحة المفاتيح أو تحريك إشارة الفأرة. كما أن شاشات الكمبيوتر الجديدة الصنع لها خاصية الإطفاء الذاتي إذا استمر هذا الوضع إلى مدة معينة نحن نحددها. وهذا جميل ولكن لسوء الحظ ولسبب غير معروف لم تقم شركة مايكروسوفت بوضع عصا الألعاب في الحسبان لذلك لو أننا برمجنا برنامج معين ليستخدم عصا الألعاب كعنصر أساسي للإدخال فإن نظام `windows` سوف يعتبر بأن المستخدم لم يقوم

باستخدام الكمبيوتر لفترة وعالية سوف يقوم بتشغيل حافظ الشاشة أو إطفاء شاشة الكمبيوتر وهذا شيء غير مستحب وقوعه ونحن في منتصف البرنامج. لذلك ولحل

هذه المشكلة سوف نقوم بإضافة الخطوات التالية:

```
case WM_SYSCOMMAND:  
  
    if((wParam&0xFFF0)==SC_SCREENSAVE ||  
       (wParam&0xFFF0)==SC_MONITORPOWER)  
  
        return 0;  
        break;
```

إلى دورة الرسائل في الوظيفة (long PASCAL WinProc) في الخطوة العاشرة من برنامج win32. الآن برنامجنا لن ينزعج ولن يتأثر بحافظ الشاشة أو خاصية الإطفاء الذاتي.

كذلك أرجو الملاحظة أن AGDX تقوم بهذه العملية لنا لذا عند استخدام المكتبة AGDX ليس هناك داعي لهذه العملية.

هوامش - هل تعلم

1- انك يجب أن تستخدم المتغيرات من نوع Static في الكائنات (Class Object) كلما أمكن ذلك لأن كل المتغيرات الأخرى المستخدمة في الكائنات يتم التعامل معها عن طريق المؤشر "this" وهو بطيء نوعاً ما.

2- استخدام التبديل المنطقي للبت "bit-shifting" أسرع من استخدام أداة الضرب أو أداة القسمة عند التعامل مع الأرقام ذات الأس اثنين. مثلاً إذا أردنا ضرب متغير معين بالعدد 16 فإننا نستخدم ما يلي:
المثال الأول:

(<< 4 المتغير) بدل ($2 \times 2 \times 2 \times 2 \times 2$ المتغير) وهذا يساوي ($16 \times$ المتغير)

المثال الثاني:

افرض أننا نريد استخدام هذه المعادلة في برنامجنا (المتغير $\times 10$) فإننا سوف نقوم أولاً بتحويل الرقم (10) إلى ما يلي ($2 + 2 \times 2 \times 2$) وهذا يساوي ($2^3 + 2^1$) بمعنى أننا نقوم بتحويل الرقم إلى شكل آخر مضروب بمضاعفات العدد (2). بعد ذلك وبكل سهولة نستطيع تحويل معادلتنا إلى: ($1 \ll$ المتغير $+ 3 \ll$ المتغير) لاحظ أن العدديين (1) و (3) هما عبارة عن أسس العدد (2).

ولكن قد تتساءل عزيزي القارئ هل هناك فعلاً سبب مقنع للقيام بذلك؟ ولنعرف الإجابة لا بد من معرفة أن أداة الضرب (أو أداة القسمة) تقوم باستخدام من 9 إلى

49 دورة حسابية للمعالج للقيام بها ، في حين أن أداة التبديل المنطقي تستخدم فقط دورتين حسابيتين وبالتالي يتضح لنا الفرق الكبير في التوقيت الذي يأخذه المعالج لحساب المعادلة السابقة في كلا الحالتين.

أما في حالة لو أردنا قسمة متغير معين برقم معين (ثمانية مثلاً) فإننا نستخدم ما يلي:

(3 >> المتغير) بدل (8 / المتغير) وهذا يساوي (2x2x2 / المتغير)

وكما نرى أن الفرق الوحيد عما سبق شرحه عند القسمة هو استخدام الاتجاه المعاكس لأداة التبديل المنطقي (>>).

3- عندما تحتاج أن تجد الموقع (X, Y) في سطح معين تستطيع أن تستخدم التبديل المنطقي بدل من الضرب كما يلي :

استخدم هذا الماكرو :

```
#define FIND_XY(x,y) ((y << 9) + (y << 7) + x)
```

بدل هذا الماكرو المساوي له :

```
#define FIND_XY(x,y) ((640 * y) + x)
```

لأنه أسرع والتعبير ((y << 9) + (y << 7)) يعني تحريك قيمة المتغير y تسع أماكن لليساو وهذا يعادل ضرب قيمة المتغير y بالرقم 512 (كما سبق وشرحنا أننا لو رفعنا العدد 2 إلى الأس 9 فإن النتيجة هي 512) مضاف إليه تحريك قيمة المتغير

y سبع أماكن لليساار وهذا يعادل ضرب قيمة المتغير y بالرقم 128 وتكون نتيجة مجموع الرقمين هو 640 مضروبة بالمتغير y.

4- بالنسبة للوظائف ذات التوقيت المهم من الأفضل استخدام المتغيرات العامة (global variables) بدل arguments لأنها تبطئ سرعة برنامجك شيئاً ما.

5- دائماً اكتب ملاحظتك وأنت تبرمج برامجك (باستخدام //) خاصة في الأماكن التي يصعب فهمها ، ربما برنامجك سهل القراءة اليوم ولكن الأمور ستتغير بعد عدة اشهر من العمل في نفس البرنامج.

الفصل الثاني

إعداد النظام *Windows 95/98/2000* لبرمجة الصوت والصورة

في هذا الفصل سنقوم بتركيب الأدوات التي سنستخدمها بشكل
تفصيلي. وسنبدأ بتركيب تكنولوجيا DirectX ثم بيئة التطوير Visual
C++ والمكتبة AGDX.

لماذا نتحدث عن تركيب الأدوات ؟

في الفصل السابق قمنا بالتحدث عن الأساسيات في لغة السي والسي المحسنة. ولكننا لم نذكر شيئاً عن كيفية برمجة الصوت الصورة. وذلك حتى تبقى الأساسيات منفصلة عن نوع البرامج التي نود صنعها. بداية من هذا الفصل سوف ندخل هذا العالم الشيق. ولكي نتلافى الكثير من المشاكل البرمجية سوف نقوم بشرح خطوات إعداد الجهاز الآلي مع الأدوات التي نحتاجها وأنظمة النوافذ بشكل مفصل. ولأن أغلب المشاكل التي تواجه أي مبتدئي في برمجة الحاسب الآلي عادة ما تكون نتيجة تركيب خاطئ أو نسيان تركيب شي معين ضروري لإنتاج البرنامج. وسوف نبدأ أولاً بالنظر لكيفية تركيب تكنولوجيا DirectX.

أولاً ماهي تكنولوجيا DirectX ؟

هي عبارة عن ملفات نضيفها لنظام التشغيل , وبتصبح جزء من النظام Windows 95 /98 /2000 ونحتاج لإضافتها بأنفسنا لكي نتأكد من أننا نستخدم آخر الإصدارات. ولكن لماذا نحتاج لوجودها أصلاً ؟ نحتاج لها لأنها عبارة عن سواقة تستغل كل مميزات الجهاز. وعن طريقها يستطيع المبرمج أن يقوم بكتابة برامج تستغل سرعة المعالجات وبطاقات العرض وغيرها من الأدوات الكثيرة والمختلفة في الأسواق بدون الخوف من أن برنامجه لن يعمل على نوع معين من تلك الأدوات.

تركيبة تكنولوجيا DirectX

مجموعة التطوير البرمجية لتكنولوجيا دايركت اكس ملحقة مع القرص المدمج لهذا الكتاب. ولكي تستطيع تركيبها تقوم أولاً بوضع القرص المدمج الملحق مع الكتاب في مشغل الأقراص في جهازك.

❖ بعد وضع القرص المدمج الملحق بالكتاب سنرى القائمة التلقائية (ملاحظة: إذا لم تظهر القائمة بشكل تلقائي شغل الملف autorun.exe مباشرة من على القرص المدمج):



من القائمة التلقائية للقرص المدمج الملحق بالكتاب نختار "صفحة التركيب".
وهي الصفحة المسؤولة عن تركيب أهم البرامج والأدوات لهذا الكتاب.

❖ بعد الضغط على صفحة التركيب سنرى النافذة التالية :



وكما نرى أننا عن طريق هذه النافذة نستطيع تركيب المكتبتين AGDX و Genesis 3D بالإضافة لتكنولوجيا DirectX وكذلك البرنامج Bryce 3D. هناك كذلك برامج أخرى موجودة على القرص المدمج نستطيع تركيبها ولكن هذه أهم البرامج لهذا الكتاب.

❖ من القائمة نختار "تركيب تكنولوجيا DirectX".

سوف نرى الشاشة التالية :



❖ من هذه النافذة كل ما علينا فعله هو الضغط على "زر التركيب".

بعدها سوف نرى الشاشة التالية :

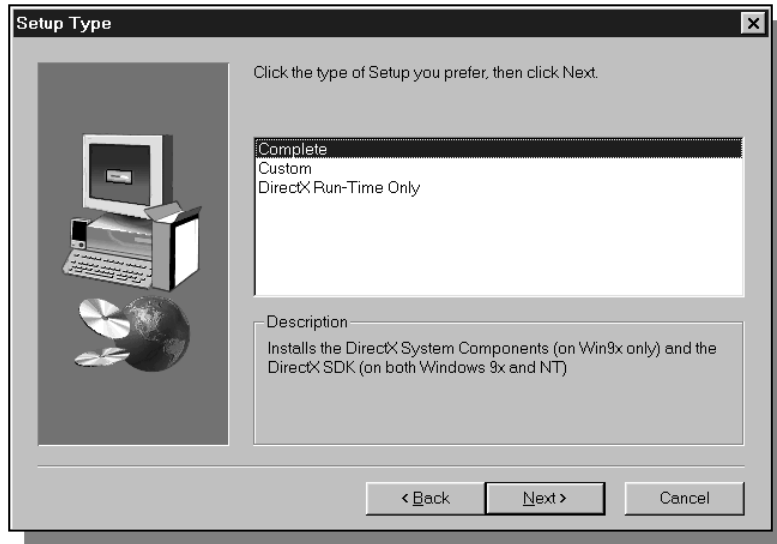


بها نافذة لإخبارنا بأننا بصدد تركيب المجموعة البرمجية لدايركت اكس على القرص الصلب بالجهاز. طبعاً نقوم بالضغط على الزر "Next" حتى نذهب للنافذة التالية.

❖ بعد ذلك سنرى نافذة بشروط الاستعمال , طبعاً يجب الموافقة على تلك الشروط الموضوعه من قبل الشركة **Microsoft** , وذلك عن طريق الضغط على الزر "Yes" إذا كنت موافق.

ملاحظة : إذا ضغط على الزر "No" الدال على عدم الموافقة سيقوم برنامج الإعداد بإنها نفسه بدون تركيب هذه التكنولوجيا.

❖ بعدها سنذهب إلى النافذة التي تلي ذلك وبها ثلاث اختيارات :



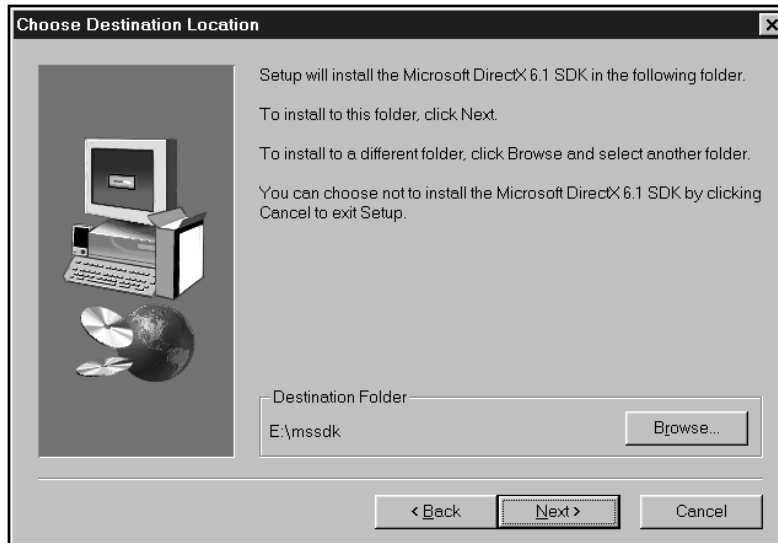
الاختيار الأول معلم عليّة تلقائياً وهو "التركيب الكامل" (Complete) ويقوم برنامج التركيب عن طريق هذا الاختيار بتركيب جميع ما نحتاجه لإنشاء واستخدام هذه التكنولوجيا على الحاسب.

بالنسبة للاختيار الثاني **Custom** فأنة وضع ليسمح لنا باختيار أجزاء معينة من هذه التكنولوجيا وعدم تركيب أجزاء أخرى.

الاختيار الثالث **DirectX Run-Time Only** فنقوم بتعليمه إذا أردنا تركيب الجزء الخاص بتشغيل برامج هذه التكنولوجيا فقط. وبما أننا في هذا الكتاب يهمننا برمجة هذه التكنولوجيا فسنقوم بإبقاء تعليم الاختيار الأول والضغط على الزر **"Next"**.

❖ في النافذة التي تلي ذلك نرى أننا أيضا أمام اختيارين أحدهما تم تعليمه لنا تلقائيا وهما **Retail** و **Debug**. نبقى الاختيار على **"Retail"** كما هو ونضغط على الزر **"Next"**

❖ في النافذة التالية :



فإن برنامج الإعداد يقوم بسؤالنا عن المكان في القرص الصلب الذي نحب تركيب هذه التكنولوجيا فيه. طبعاً يقوم بإعطائك مكان تلقائي في حالتي المكان هو :

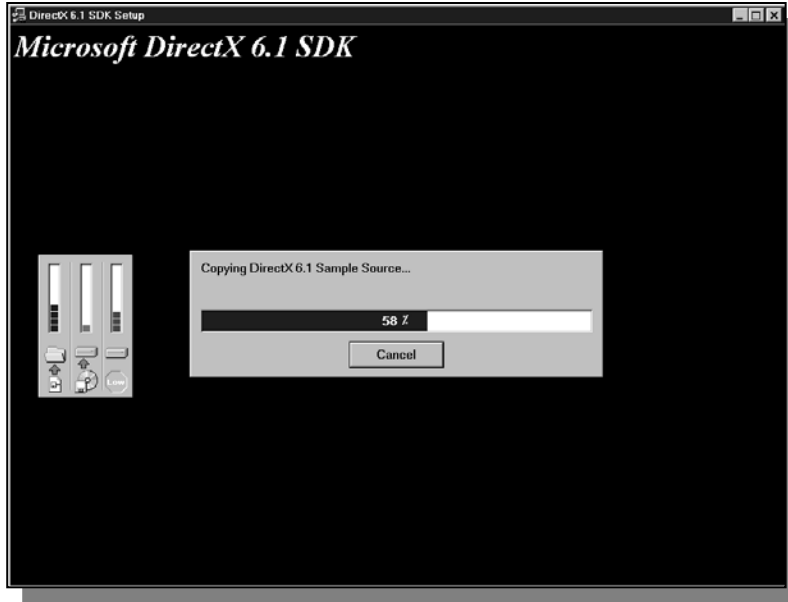
E:\mssdk

هذا لأن نظام ويندوز على جهازي يوجد في القرص " E " (عندي أكثر من قرص صلب). إذا لم تمنع المكان التلقائي الذي يختاره لك تضغط على الزر "Next". إذا أحببت تغيير مكان التركيب تضغط على الزر "Brows" وتقوم باختيار المكان المناسب , ثم بعد ذلك تضغط على الزر "Next".

❖ في النافذة التي تلي ذلك تقوم بسؤالنا عن المكان في قائمة البرامج الذي نحب فيه تركيب هذه التكنولوجيا. سوف نقوم بقبول المعطيات التلقائية وهي إنشاء ملف خاص في قائمة البرامج اسمه "Microsoft DirectX 6.1 SDK" عن طريق الضغط على الزر "Next".

❖ في هذه النافذة يقوم الكمبيوتر بإخبارك انه على استعداد الآن لتركيب هذه التكنولوجيا وإذا أردنا الرجوع إلى أي نافذة سابقة نستطيع الضغط على الزر "Back" للرجوع للوراء وتغيير ما نراه مناسب. سنقوم باختيار "Next" لأننا متأكدين من اختياراتنا السابقة.

❖ بعدها سيبدأ برنامج الإعداد في تركيب هذه التكنولوجيا ونرى الشاشة التالية:



عند الانتهاء من التركيب ستظهر لنا الرسالة التالية :

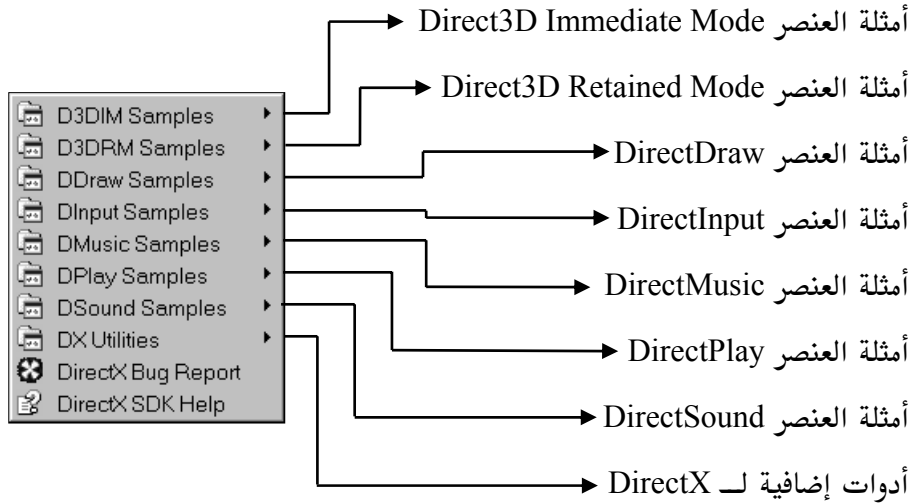


والتي تخبرنا بأن برنامج الإعداد قام بحفظ الملف DXSETENV.BAT على القرص الصلب في المكان الذي ركبنا فيه هذه التكنولوجيا ومن ضمن ملفات **.bin**. ونستطيع عن طريق هذا الملف أن نقوم بإنتاج برامجنا من إشارة الـ **Dos**, وبما أننا في هذا الكتاب لن نستخدم هذه الطريقة القديمة لإنتاج البرامج فإننا لا نهتم كثيرا لهذه الرسالة وسنقوم بالضغط على الزر "OK".

❖ وفي النافذة الأخيرة نقوم بالضغط على الزر "Finish"

على ماذا تحتوي تكنولوجيا DirectX ؟

الآن بعد أن ركبنا المجموعة البرمجية على جهازنا لنقم سوف ننظر إلى محتوياتها. من قائمة البداية "Start" نختار قائمة البرامج "Programs" , منها نختار المجموعة البرمجية لدايركت اكس "Microsoft DirectX 6.1 SDK" وفيها نرى القائمة التالية :



كما ترى أن هناك عشرات الأمثلة لكل عنصر من عناصر تكنولوجيا DirectX , وجميعها تأتي مع الشفرة البرمجية لها. لن نتمكن في هذا الكتاب من شرح كل مثال من تلك الأمثلة. في المقابل سوف نستخدم المكتبة AGDX وهي طبقة إضافية توضع فوق مكتبة DirectX لتجعل التعامل معها أكثر سهولة وسلاسة. كما سوف ترى في الأمثلة في الفصول القادمة ما سيجعلك تتفادى التعامل المباشر مع مكتبة DirectX. في الجزء الثاني سوف نشرح الطريقة المباشرة لعمل العنصر DirectDraw وهو العنصر المسؤول عن الرسوم التي نراها على الشاشة وكذلك العنصر Direct3D وهو العنصر المسؤول عن الأبعاد الثلاثية. شرح هذين العنصرين في الجزء الثاني من الكتاب سيجعل من أمثلة دايركت اكس سهلة الفهم لأن جميعها تنطبق عليها القوانين نفسها. لكن هل نحتاج فعلا لمعرفة كيفية التعامل المباشر مع هذه التكنولوجيا ؟ للكثيرين الجواب لا , لأن كل ما يهمنا هو إنتاج برامجنا في اقصر وقت وبأفضل الإمكانيات. أما للبعض الآخر فمعرفة التعامل المباشر مع هذه التكنولوجيا سوف يسمح بالاستفادة منها أكثر كما أنها تعطيتهم القدرة على تعديل المكتبة AGDX لتنماشى مع متطلباتهم الخاصة. في كلتا الحالتين سنحاول في هذا الكتاب أن نعطي فكرة واضحة عن كيفية الاستفادة من هذه التكنولوجيا المذهلة وتسخيرها لمتطلباتنا.

ما هي بيئة التطوير 6 Visual C++ ؟

لكي نستطيع إنتاج برامجنا سنحتاج لهذه البيئة. وهي عبارة عن عدة برامج تجمعها واجهة نستطيع من خلالها تحويل شفرتنا البرمجية والتي سنكتبها بلغة السي المحسنة C++ إلى برامج نستطيع تشغيلها على حواسبننا الآلية وبلغة الكمبيوتر.

مثلا نحن عندما نقوم بكتابة شفرة برامجنا فإننا نكتبها على ملف أو عدة ملفات. بعد الانتهاء من كتابتها نقوم باستخدام برنامج يسمى المصرف "Compiler" بقراءتها والتأكد أنها خالية من الأخطاء. طبعاً الأخطاء المطبعية والأخطاء البرمجية. بعدها نستخدم برنامج آخر يسمى "الرابط" (Linker) بربط المكتبات البرمجية التي استخدمناها في شفرة البرنامج (مثل مكتبة DirectX والمكتبة AGDX) مع الشفرة البرمجية وإعطائنا النتيجة الأخيرة (وهي في حالتنا ملف ينتهي بالأحرف exe) والتي نستطيع تشغيله مثل أي برنامج آخر.

الجميل في بيئة التطوير Visual C++ أن كل هذه العملية تتم بشكل تلقائي وكل ما علينا فعله هو التأكد بأننا قمنا بإعداد البرنامج بالشكل الصحيح. وهذا ما سنقوم به في الخطوة القادمة.

هذا طبعاً بالإضافة أن بيئة التطوير توفر لنا الأدوات المناسبة لكتابة وتنقيح برامجنا من الأخطاء. وتسمى هذه العملية بـ Debug أي إزالة الحشرات "الحشرات يقصد بها الأخطاء في عالم البرمجة".

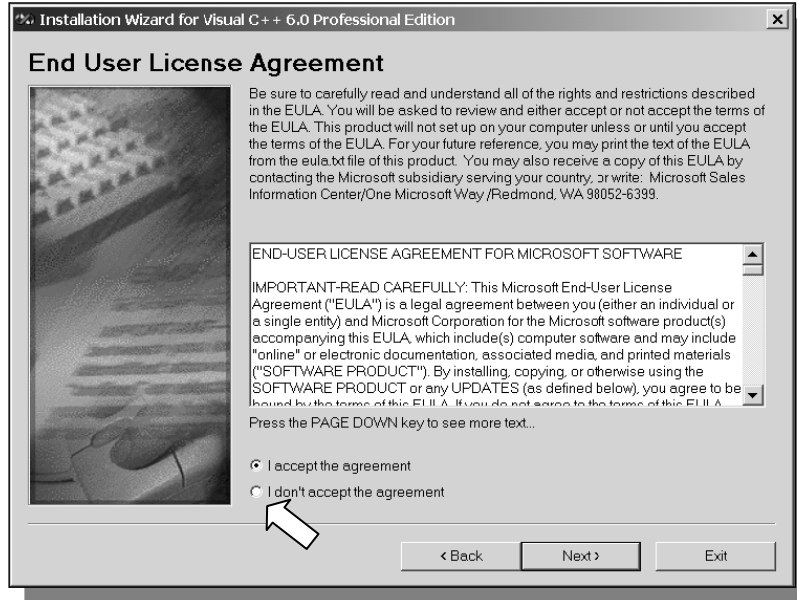
تَرْكِيْبُ بِيئَةِ التَّطْوِيرِ 6 Visual C++ ؟

تأتي بيئة التطوير على قرص مدمج . يجب أن تحصل عليه بطريق منفصل لأنه لا يأتي من ضمن هذا الكتاب , إذا كنت لا تعرف من أين تحصل عليه تستطيع أن تقوم بطلبه مباشرة من شركة www.Microsoft.com . يأتي كذلك كجزء من مجموعة التطوير **Microsoft Visual Studio 6.0** والتي تضم مجموعة برامج وهي **Visual C++ 6** و **Visual J++ 6** و **Visual Basic 6** و **Visual FoxPro 6** و **Visual InterDev 6** وأخيراً **Visual SourceSafe 6**.

لسبب معين فإن طلب نظام التطوير مباشرة من شركة **Microsoft** غالي نوعاً ما. ولكن هناك عشرات الأماكن التي تبيعه بسعر أرخص بكثير. وإذا كنت طالب في مدرسة أو كلية أو جامعة فيحق لك أن تحصل عليه بسعر خاص للطلبة. لمعلومات أكثر راجع موقع شركة **Microsoft.com** على الإنترنت. أو يستحسن أن تبحث في محلات الكمبيوتر التي بقربك لأسعار أنا متأكد أنها أرخص بكثير من أسعار شركة **Microsoft**.

❖ بعد أن نضع القرص المدمج في مشغل الأقراص سوف نرى شاشة الإعداد (إذا لم تشتغل شاشة الإعداد بشكل تلقائي , يجب عليك أن تشتغل الملف **Setup.exe** بنفسك من على القرص المدمج). من على الشاشة الإعداد نضغط على الزر **Next**.

❖ في النافذة الجديدة نرى ترخيص الاستعمال :



طبعاً إذا أردنا استخدام هذا البرنامج نقوم بقبول ترخيص الاستعمال عن طريق

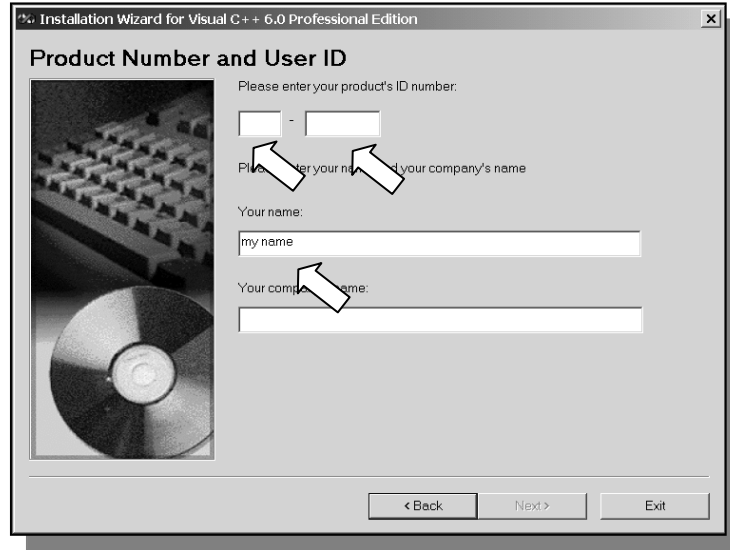
● I accept the agreement اختيار القبول

ثم الضغط على الزر Next.

❖ في النافذة التي تلي ذلك نقوم بإدخال الاسم ورقم البرنامج (كل برنامج

لشركة مايكروسوفت يأتي مع رقم يكون مكتوب عادة على الغطاء الخارجي

للقرص المدمج).



بعد تعبئة المعلومات المطلوبة نقوم بالضغط على الزر **Next**.
ملاحظة : يجب التأكد من إدخال الرقم الصحيح وإلا فأنتك لن تستطيع الضغط
على الزر **Next**.



❖ بعد ذلك
سنرى النافذة
التالية :

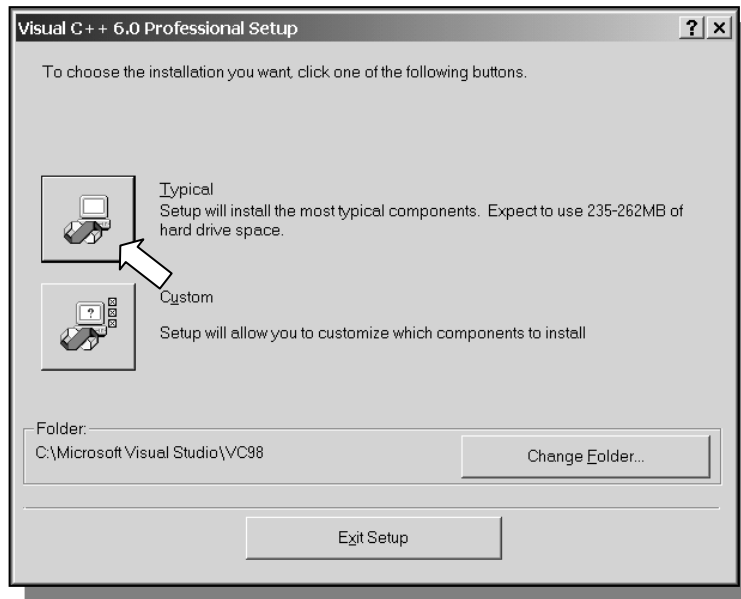
والتي تسألنا عن المكان الذي نود فيه تركيب البرنامج ،

وتعطينا اقتراح لمكان على القرص المدمج وحجم القرص وكذلك الحجم الذي يحتاجه البرنامج لتركيب بيئة التطوير. إذا كنت موافق على كل الاقتراحات التي يعطيك إياها برنامج الإعداد فقم بالضغط على الزر Next.

❖ في النافذة التي تلي ذلك نقوم بالضغط على الزر Continue

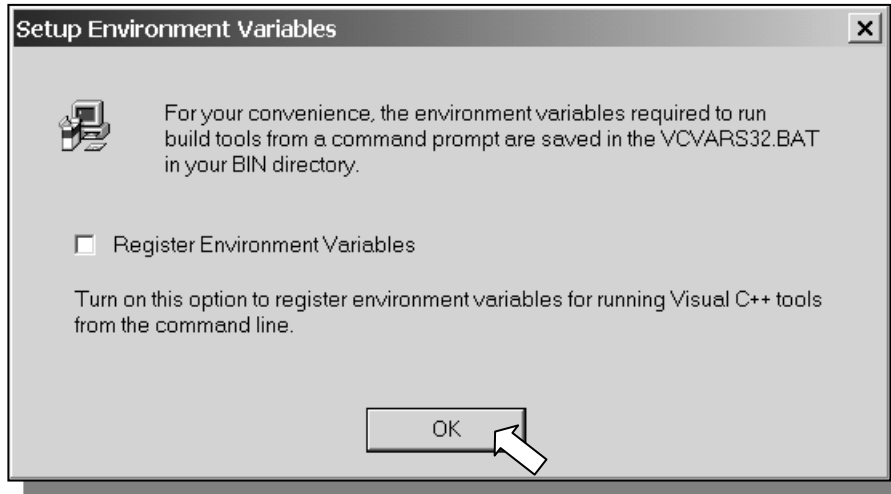
❖ في النافذة التي تلي ذلك نضغط على الزر OK

❖ بعد ذلك نرى النافذة التالية :



في هذه النافذة لنا اختاران : الأول Tycipal ويعني بتركيب جميع المواد اللازمة والضرورية لبيئة التطوير و الاختيار الثاني Custom ويعني أننا نريد أن نحدد نوعية التركيب بأنفسنا. سنقوم هنا بالضغط على الاختيار الأول.

❖ بعد ذلك نرى النافذة التالية :

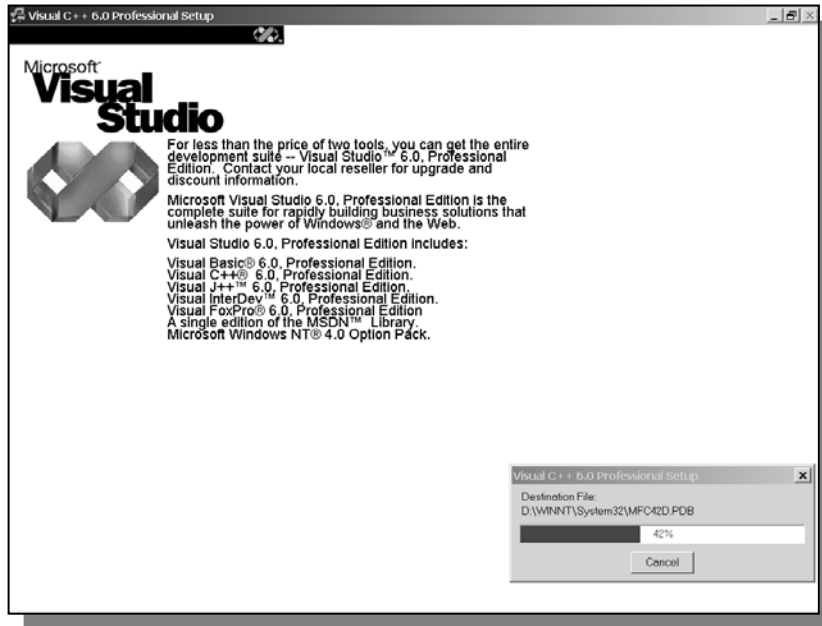


والتي تخبرنا بأن برنامج الإعداد قام بحفظ الملف VCVARS32.BAT على القرص الصلب في المكان الذي ركبنا فيه بيئة التطوير ومن ضمن ملفات bin. ونستطيع عن طريق هذا الملف أن نقوم بإنتاج برامجنا من إشارة الـ Dos, وبما أننا في هذا الكتاب لن نستخدم هذه الطريقة القديمة لإنتاج البرامج فإننا لا نهتم كثيرا لهذه الرسالة وسنقوم بالضغط على الزر "OK".

نضغط على الزر OK

لو تتذكر عند تركيب تكنولوجيا دايركت اكس وجه إلينا نفس السؤال وذلك يعني أننا باستطاعتنا استخدام بيئة التطوير مباشرة من إشارة الـ **Dos** لأنها الطريقة المتبعة في الماضي لإنتاج البرامج ولأن الكثير قد تعود عليها ووضعت هنا كإضافة اختيارية.

❖ بعد ذلك سيبدأ برنامج الإعداد في تركيب بيئة التطوير.



❖ وفي النافذة الأخيرة نقوم بالضغط على الزر OK مرة أخرى.

❖ وأخيرا سوف نرى النافذة التالية :



والتي يخبرنا فيها برنامج الإعداد أن التركيب تم بنجاح ونستطيع الآن أن نبدأ باستخدام بيئة التطوير. طبعاً نضغط على الزر OK

سيقوم كذلك برنامج الإعداد بسؤالنا فيما إذا أردنا تركيب برامج أخرى نحن لسنا بحاجة إليها لذلك نقوم بالخروج.

هل نحن جاهزون لاستخدام بيئة التطوير ؟

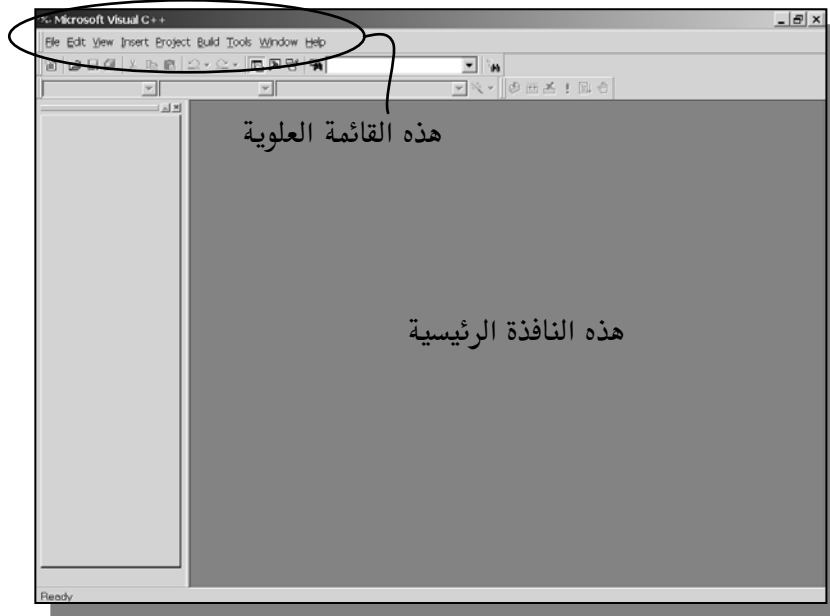
ليس تماماً , لأننا سنقوم باستخدام مكتبات تكنولوجيا **DirectX** وكذلك المكتبة **AGDX** ثم المكتبة **Genesis 3D**. هناك بعض الخطوات التي يجب القيام بها لربط هذه المكتبات قبل أن نكون مستعدين لاستخدامها مع بيئة التطوير.

- نقوم أولاً بتشغيل برنامج التطوير Visual C++ عن طريق اختياره من قائمة البرامج:

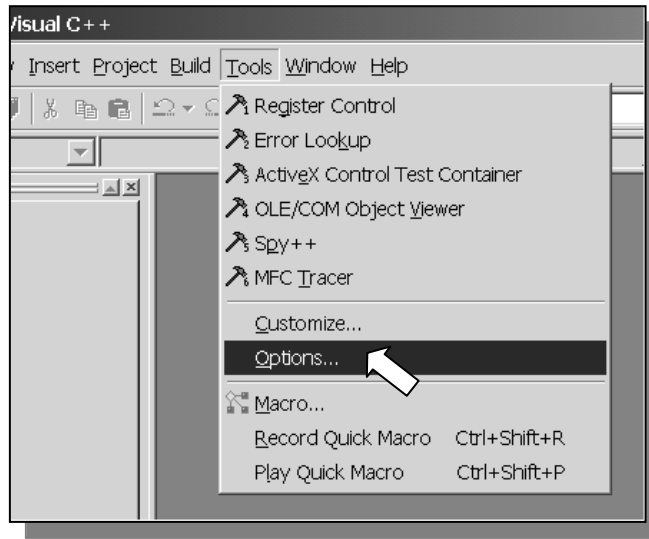


هناك أيضاً مجموعة من الأدوات التي تأتي مع بيئة التطوير . ونحن في هذا الكتاب لن نهتم لها كثيراً ولكن ذلك لا يمنع من أن تتطلع عليها بنفسك وترى إذا كنت تستطيع الاستفادة منها.

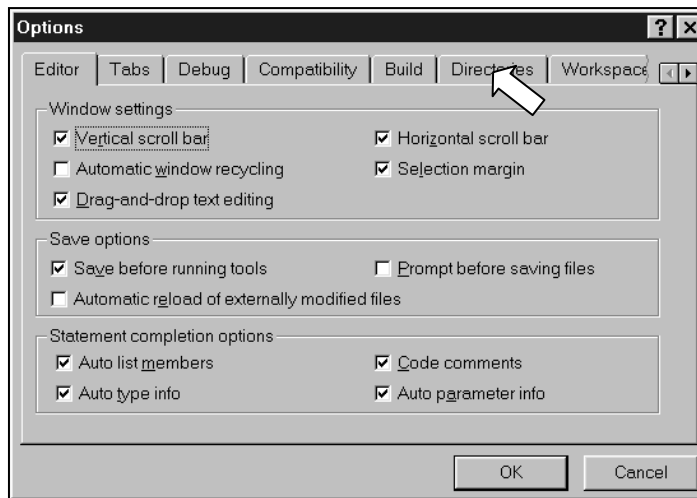
بعد لحظات سنرى الشاشة الرئيسية للبرنامج وتبدو كما يلي :



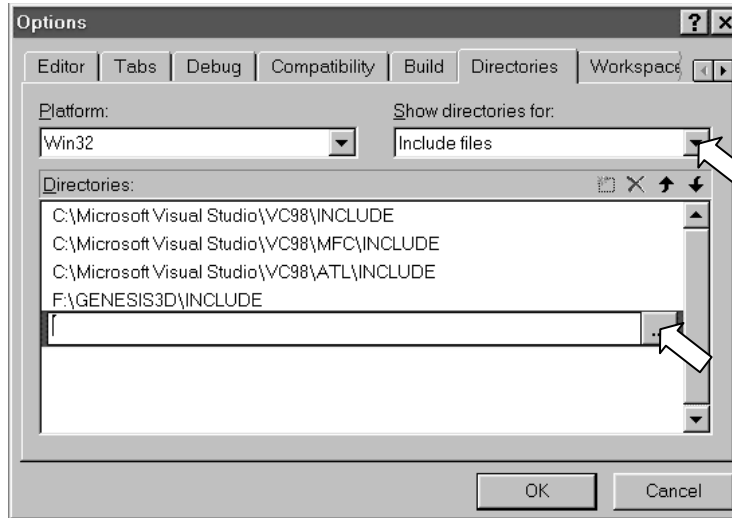
- بعد ذلك من القائمة العلوية نختار **Options...** من الأمر **Tools**:



- سوف نرى النافذة التالية:



نختار **Directories** من القائمة العلوية. سوف نرى النافذة التالية :



من خلال هذه النافذة نخبر بيئة التطوير بالمكان الذي توجد به ملفات التعريف التابعة للمكتبات التي سوف نستخدمها (**Include files**) وهي الملفات التي تنتهي بالحرف **h**. كذلك نخبر المكتبة عن مكان وجود ملفات المكتبات التي سوف نستخدمها (**Library files**) وهي الملفات التي تنتهي بالأحرف **Lib**. ونحن في هذا الكتاب سوف نستخدم ثلاث أنواع من المكتبات : مكتبات تكنولوجيا **DirectX** والمكتبة **AGDX** والمكتبة **Genesis 3D**.

مثلا في الجزء الأول من هذا الفصل شرحنا طريقة تركيب تكنولوجيا **DirectX** وقمنا بإخبار برنامج الإعداد عن المكان الذي يجب أن تركيب فيه هذه التكنولوجيا. هنا نقوم بإخبار بيئة التطوير عن نفس المكان الذي يحتوي على الملفات المطلوبة. مما يعني انك يجب أن تركيب المكتبات المعنية قبل القيام

بهذه الخطوة وإلا فأنت لن تجد ملفاتك على قرصك الصلب. وهذه الخطوة مهمة جدا وبدونها فإن برامجنا لن تعمل.

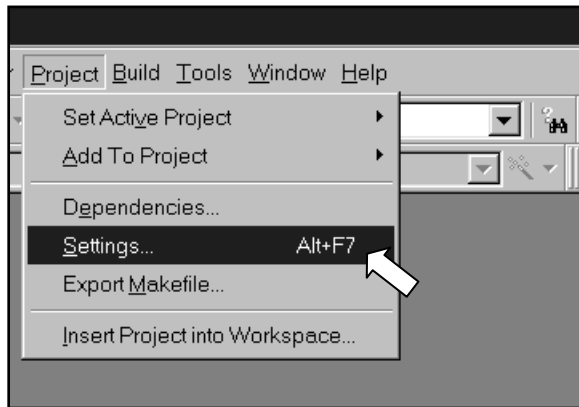
عفوا أنا لا زلت جديدا على هذه البيئة ولا ادري ماذا تقصد بملفات التعريف وملفات المكتبات ولماذا نحتاج إخبار بيئة التطوير بها ؟



المكتبات هي عبارة عن ملفات من نوعين : نوع ينتهي بالحرف **h** مثلا **AGDX.h** وتسمى الملفات التعريفية , ونوع ينتهي بالأحرف **lib** مثلا **AGDX.lib** وتسمى ملفات المكتبات. عن طريق هذين النوعين من الملفات نستطيع استغلال مميزات كل مكتبة نستخدمها (هناك أنواع أخرى لن نهتم بها في هذه المرحلة). ولذلك فنحن نحتاج لأخبار بيئة التطوير عن مكان تلك الملفات والتي من الفروض أن تكون قد ركبت على القرص المدمج عند تركيب مكتباتها. وما قمنا به في الخطوة السابقة هو إخبار بيئة التطوير عن مكان وجود هذين النوعين من الملفات لكل مكتبة نحن نستخدمها.

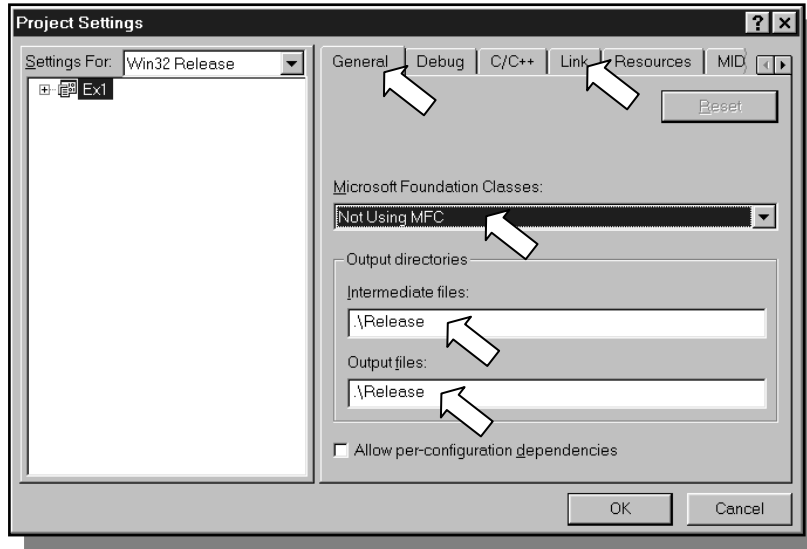
هل هذا كل شيء ؟

نعم ولا , كما أوضحنا سابقاً أن كل أمثلة هذا الكتاب ستجدها على القرص المدمج الملحق , وكل ما عليك فعله هو تعبئة المثال المطلوب إلى بيئة التطوير وسوف يعمل بدون أن تحتاج للقيام بالخطوات التي سنشرحها بعد قليل. ولكن ماذا لو أردت أن تبدأ برنامجك الخاص بدون استخدام أيّاً من الأمثلة الموضحة في هذا الكتاب. في هذه الحالة ستضطر أن تقوم بنفسك ببعض الخطوات البسيطة لربط ملفات المكتبات بالمشروع , لماذا ؟ لان هناك برنامج يسمى "الرابط" سبق وذكرناه في هذا الفصل عندما اجبنا عن السؤال "ما هي بيئة التطوير **Visual C++ 6** ؟" يقوم بربط ملفات المكتبات بالشفرة البرمجية التي



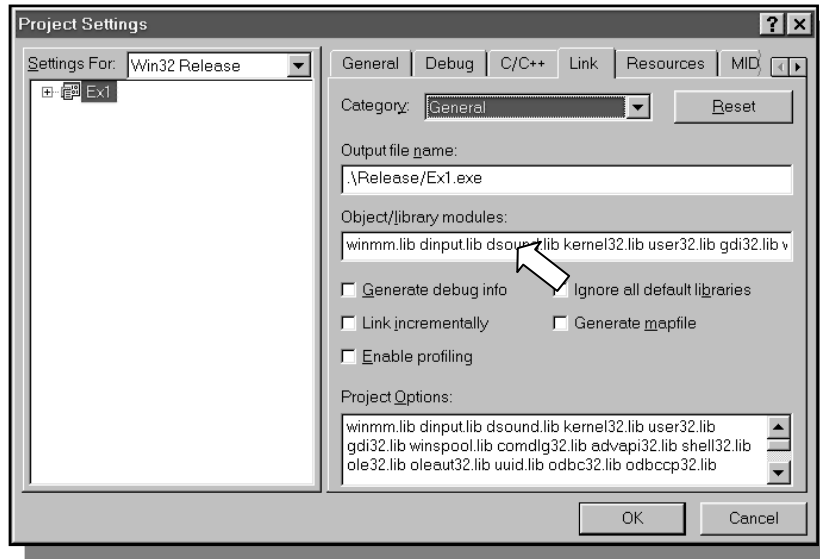
نكتبها ويقوم بالبحث عن ملفات المكتبات التي نختبره باستخدامها. ولكي تكون هذه الخطوة صحيحة نقوم بما يلي :

- ❖ (طبعاً على افتراض انك بدأت مشروع جديد) نختار **Settings...** من الأمر **Project** في القائمة العلوية لبيئة التطوير.
- ❖ بعد ذلك نرى النافذة التالية :



هذه النافذة هي المسؤولة عن كل ما يحصل في مشروع (**Project**) جديد. وإذا كنت لا تعرف ما هو المشروع وما فائدته إقرأ الفصل القادم "برمجة النوافذ". هنا سنقوم فقط بالتأكد أن كل شيء على ما يرام. كما ترى في أعلى هذه النافذة أن هناك أكثر من اختيار وأننا نحن الآن في الاختيار "General" ويعني "عام" وفي هذا الاختيار سنقوم في المستطيل الأول باختيار "Not Using MFC" ويعني ان مشروعنا لا يستخدم مكتبات MFC , وفي المستطيل أدناه ذلك نكتب "Release" . ويعني أن مشروعنا سيجد الملفات التي يحتاجها (مثل الصور والأصوات ...الخ) في الملف Release وكذلك في المستطيل الأسفل نكتب نفس الشيء ويعني أننا نريد من المصرف عند إنتاج ملف برنامجنا exe أن يضعه في نفس المكان.

❖ بعد ذلك وفي نفس النافذة السابقة نضغط على الأمر **Link** ويعني "ربط" في القائمة العلوية عندها سنرى النافذة التالية :



وخيرا وصلنا للنافذة التي ينظر إليها برنامج "الرابط" وفيها نخبره بالمكتبات التي استعملناها. طبعاً نستطيع ربط المكتبات التي لم نستعملها كذلك ولكن هذا سيزيد من حجم الملف الناتج.

افرض مثلاً أننا نريد استخدام تكنولوجيا دايركت اكس فقط في أحد مشروعاتنا عندها في المستطيل المكتوب أعلاه "Object/library modules:" سنقوم

بإضافة الملفات التالية:

- ddraw.lib (العنصر DirectDraw)
- dsound.lib (العنصر DirectSound)

- `dinput.lib` (العنصر **DirectInput**)
- `dxguid.lib` (المكتبة المسؤولة عن الأخطاء التي قد تحصل في عناصر **DirectX** مهم جدا إرفاقها عند برمجة هذه التكنولوجيا)
- `d3drm.lib` (العنصر **Direct3D Rational Mode**)
- `d3dim.lib` (العنصر **Direct3D Immediate Mode**)
- `winmm.lib` (أحد مكتبات `win32` والتي تأتي مع بيئة التطوير وهو المسؤول عن تشغيل ملفات الموسيقى **midi** التي سوف نستخدمها لاحقا)
- `vfw32.lib` (أيضاً أحد مكتبات `win32` والتي تأتي مع بيئة التطوير وهو المسؤول عن تشغيل ملفات الفيديو **avi** التي سوف نستخدمها لاحقا)


ونكتبها بشكل متتالي ونترك بين كل منها فراغ كما يلي :

`ddraw.lib dsound.lib dinput.lib d3drm.lib winmm.lib`

طبعا ليس شرطا إلحاق جميع تلك المكتبات في كل مشروع جديد (أو جميع عناصر **DirectX**) , ولكن فقط ما نحتاج. مثلا لإضافة المكتبة **AGDX** فإننا نضيف الملف **AGDX.lib** ولكي نضيف المكتبة **Genesis3D** فإننا نضيف الملف **Genesis.lib** وهكذا.

إذا وصلت لهذه النقطة وقمت بتنفيذ جميع الخطوات السابقة بنجاح فأنت الآن مستعد لاستخدام بيئة التطوير.

وأخيرا نضغط على الزر "OK".

عفوا , عندي سؤال في هذا الموضوع : ماذا يحصل لو أننا نسينا إلحاق أحد تلك المكتبات المطلوبة في مشروعنا ؟ 

سوف تحترق شاشة العرض لديك !!! (لا لا تخف , هذه مجرد مزحة فقط ☺)
الجميل في عالم البرمجة أننا عندما نخطي فإن أسوء ما سيحصل هو إطفاء الجهاز وإعادة تشغيله. لذلك لا تقلق بشأن القيام بخطأ ما. الآن نعود للإجابة عن سؤالك , إذا نسيت إلحاق مكتبة معينة أنت استخدمتها في الشفرة البرمجية سوف يعطيك برنامج "الرابط" الجملة التالية (تسبقها وتليها عدة جمل أخرى) :

unresolved external

وتعني أن برنامج "الرابط" لم يجد تلك الأوامر التي تتحدث عنها في شيفرتك البرمجية. طبعا هذا لكل مكتبة أنت نسيت ربطها وكل ما عليك فعله هو إضافة تلك المكتبة كما فعلنا سابقا حتى يجدها برنامج الرابط . وإذا واجهتك نفس الرسالة مرة أخرى فهذا يعني أن هناك مكتبة أخرى أيضا نسيت ربطها , وهكذا يجب أن تتأكد من ربط جميع المكتبات التي استخدمتها.

تركيب المكتبة AGDX

وأخيراً سنقوم بشرح طريقة تركيب المكتبة AGDX :

- ❖ بعد وضع القرص المدمج في الكتاب سنرى القائمة التلقائية (ملاحظة : إذا لم تظهر القائمة بشكل تلقائي شغل الملف autorun.exe مباشرة من على القرص المدمج) :



من القائمة التلقائية للقرص المدمج الملحق بالكتاب نختار "صفحة التركيب". وهي الصفحة المسؤولة عن تركيب أهم البرامج والأدوات لهذا الكتاب.

❖ بعد الضغط على صفحة التركيب سنرى النافذة التالية :



وكما فعلنا فيما قبل مع تركيب تكنولوجيا DirectX نقوم هنا باختيار "تركيب

المكتبة
"AGDX"

❖ بعدها
سوف نرى
الشاشة
التالية :



بعدها سيبدأ برنامج التركيب في العمل.

❖ اضغط على الزر Next حتى ترى الزر Finish اضغط عليه وسوف تتركب المكتبة على المسار : C:\AGDX طبعا تستطيع تغيير مكانها على القرص أن أردت.

وبهذا نكون قد ركبنا أهم البرامج لهذا الكتاب. البرامج المتبقية سوف تجدها على الملف Extra في القرص المدمج وتستطيع الوصول إليه عن طريق اختيار " معلومات إضافية" من القائمة الرئيسية على القرص المدمج.

برمجة الأبعاد الثنائية والثلاثية
للكمبيوتر
2D & 3D

مرحبا بكم في قائمة القرص المدمج التابع للكتاب. في هذا القرص ستجد جميع الأدوات والبرامج التي تحدثنا عنها وأمثلة كثيرة.

صفحة التزكيب

صفحة الأمثلة

معلومات إضافية

خروج

تعليم : النوافذ والنشر غير مسؤولين عن أي خطأ في البرنامج يحصل من جراء استعمال الكتاب أو القرص. إذا لم توافق على الشراء الرجاء عدم استعمال القرص.

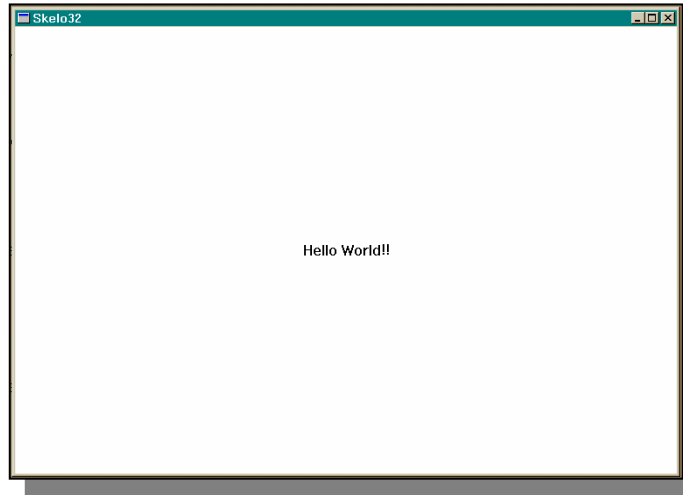
الفصل الثالث

Win32

برمجة

Win 32

كل برنامج يعمل على نظام النوافذ يستخدم واجهة برمجية يطلق عليها **Win 32**. سوف نلخص خطوات إنشاء برنامج كامل باستخدام هذه الواجهة في عشر خطوات ، وينطبق هذا الشرح على جميع برامج ويندوز سواء أكانت تستخدم DirectX أم لا. لذلك فإن معرفة كيفية عمل هذا البرنامج مهمة جدا وسوف توفر عليك الكثير من علامات الاستفهام في برمجه أنظمة ويندوز وفي فهم بقية أمثلة الكتاب. بعد ذلك سنتعرف على ميكانيكية يطلق عليها **Win32 Application** تقوم بكتابة هذا البرنامج لنا بشكل أوتوماتيكي.



```
                                //Win32.CPP
#include <windows.h>
#include <windowsx.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain (
HINSTANCE hInstance, // handle to current instance
HINSTANCE hPrevInstance, // handle to previous instance
PSTR szCmdLine, // pointer to command line
int iCmdShow // show state of window
)
{
// STEP #1 - Set up your variables -----

WNDCLASSEX wndclass;
MSG msg;
HWND hwnd
static char szAppName[] = "First Windows Program";

// STEP #2 - Set Elements of a "WNDCLASSEX" structure

wndclass.cbSize = sizeof (wndclass);
wndclass.style = CS_HREDRAW | CS_VREDRAW;
wndclass.lpfnWndProc = WndProc;
wndclass.cbClsExtra = 0;
wndclass.cbWndExtra = 0;
wndclass.hInstance = hInstance;
wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION);
wndclass.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
wndclass.hCursor = LoadCursor (NULL, IDC_ARROW);
wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
wndclass.lpszMenuName = NULL;
wndclass.lpszClassName = szAppName;

// STEP #3 - Register the "WNDCLASSEX" structure with the operating
// system

if (!(RegisterClassEx (&wndclass)))
return FALSE;
```

```
// STEP #4 - Now you can actually create the window

hwnd = CreateWindowEx (
    WS_EX_APPWINDOW,
    szAppName,
    "Skelo32",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    NULL,
    NULL,
    hInstance,
    NULL);

// STEP #5 - Display the window

ShowWindow (hwnd, iCmdShow);

// STEP #6 - Force an update to the window

UpdateWindow (hwnd);

// Step #7 - The WinMain function enters an infinite loop which is referred
// to as a windows "message pump"

while (TRUE)
{
    if (!GetMessage (&msg, NULL, 0, 0))
        return msg.wParam;
    TranslateMessage (&msg);
    DispatchMessage (&msg);
}

// Step #8 - The windows callback function "WinProc"

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM
wParam, LPARAM lParam)
{
```

```
// Step # 9 WinProc - Define variables
HDC hdc;
PAINTSTRUCT ps;
RECT rect;
static BOOL fFirstPaint = TRUE;

// Step # 10 WinProc - The switch statement for handling windows messages

switch (iMsg)
{
case WM_KEYDOWN :
switch (wParam)
{
case VK_ESCAPE:
case VK_F12:
PostMessage (hwnd, WM_CLOSE, 0, 0);
return 0;
}
return 0;
case WM_PAINT :
hdc = BeginPaint (hwnd, &ps);

GetClientRect (hwnd, &rect);

DrawText (hdc, "Hello World!!", -1, &rect, DT_SINGLELINE |
DT_CENTER | DT_VCENTER);
EndPaint (hwnd, &ps);

if (fFirstPaint)
{
ShowCursor (TRUE);
fFirstPaint = FALSE;
}
return 0;

case WM_DESTROY :
PostQuitMessage (0);
return 0;
}

return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

ونشرح هذا البرنامج حسب ترتيبه من بدايته إلى نهايته كما يظهر في الملف المكتوب فيه (البرنامج win32 موجود على القرص المدمج مع ملف بيئة التطوير وكل ما عليك فعله هو تعبئته في Visual C++6 وتكون جاهزا للعب معه).

قبل الدخول في خطوات البرنامج نقوم أولاً باستخدام الأمر include مع الملفين <windows.h> و <windowsx.h> وذلك لاحتوائهما على تعاريف الأوامر المستخدمة وهما جزء من win 32 هذا كل ما نحتاج معرفته في هذين الملفين. ثم كتابة التعريف التالي:

```
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM,
                           LPARAM) ;
```

ويستحسن أن نضع هذا التعريف في ملف خاص نستخدمه للتعريف المستخدمة في برنامجنا (مثلاً `hdrs.h`) ولكن لقصد التوضيح فلا بأس من كتابته في البرنامج الرئيسي. وهو تعريف وظيفة Windows Callback Function (راجع الخطوة الثامنة)

- هنا سوف نلاحظ الاختلاف في برمجة ويندوز عن برمجة Dos ، في برمجة Dos نحن نعرف أن بداية البرنامج تكون من "main" ولكن في نظام ويندوز فإن بداية البرنامج تكون من "WinMain" كما يلي:

```
int WINAPI WinMain (
    HINSTANCE hInstance,           // handle to current instance
    HINSTANCE hPrevInstance,      // handle to previous instance
    PSTR szCmdLine,               // pointer to command line

    int iCmdShow                  // show state of window
)
{
```

الخطوة الأولى

نقوم بتعريف المتغيرات المستخدمة

- يحمل هذا المتغير جميع المعلومات المطلوبة عن كائن النافذة :
WNDCLASSEX wndclass;
- يقوم هذا المتغير بحمل جميع رسائل ويندوز:
MSG msg;
- هذا المتغير عبارة عن رقم برنامجنا بحيث نتعرف عليه ويندوز :
HWND hwnd;
- هذا المتغير عبارة عن مجموعة أحرف تكون في مجموعها اسم البرنامج:
static char szAppName[] = " First Windows Program";

الخطوة الثانية

"WNDCLASSEX"

```
wndclass.cbSize = sizeof (wndclass);
wndclass.style = CS_HREDRAW | CS_VREDRAW;
wndclass.lpszClassName = szAppName; ← شرح هذا الأمر في الخطوة الثامنة
```

```
wndclass.cbClsExtra = 0;
wndclass.cbWndExtra = 0;
wndclass.hInstance = hInstance;
wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION);
wndclass.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
wndclass.hCursor = LoadCursor (NULL, IDC_ARROW);
wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
wndclass.lpszMenuName = NULL;
wndclass.lpszClassName = szAppName;
```

الخطوة الثالثة

في هذه الخطوة نقوم بتسجيل "WNDCLASSEX" مع نظام التشغيل

```
if (!(RegisterClassEx (&wndclass)))
    return FALSE;
```

الخطوة الرابعة

الآن نستطيع أن نقوم بإنشاء النافذة للبرنامج

```
hwnd = CreateWindowEx (
    • نضع نوع النافذ Extended Window Style :
    WS_EX_APPWINDOW,
    • أسم كائن النافذة:
    szAppName,
    • الاسم الظاهر في أعلى النافذة :
    "Skelo32",
    • صفات النافذة :
    WS_OVERLAPPEDWINDOW,
    • القيمة المبدئية للموقع x :
    CW_USEDEFAULT,
    • القيمة المبدئية للموقع y :
    CW_USEDEFAULT,
    • القيمة المبدئية لحجم x :
    CW_USEDEFAULT,
    • القيمة المبدئية لحجم y :
```

```

NULL,
NULL,
hInstance,      رقم برنامجنا :
NULL);

```

الخطوة الخامسة

إظهار النافذة

```
ShowWindow (hwnd, iCmdShow);
```

الخطوة السادسة

نقوم بفرض تجديد النافذة

```
UpdateWindow (hwnd);
```

الخطوة السابعة

تقوم "WinMain" بالدخول في دورة لانتهائية يطلق عليها دورة رسائل ويندوز بحيث يقوم برنامجنا بمراقبة الرسائل مثل حركة الفأرة و أزرار لوحة المفاتيح (أنظر إلى الشكل (1))

```

while (TRUE)
{
    if (!GetMessage (&msg, NULL, 0, 0))
        return msg.wParam;
    TranslateMessage (&msg);
    DispatchMessage (&msg);
}

```

• هنا نهاية "WinMain"

```

}

```


الخطوة الثامنة- (تشمل هذه الخطوة الخطوتين الأخيرتين)

في هذه الخطوة سوف نرى وظيفة الاستدعاء لهذا البرنامج (The windows)

ويطلق عليها "WinProc" (callback function)

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg,
WPARAM wParam, LPARAM lParam)
{
```

الخطوة التاسعة

```
HDC          hdc;      هذا المتغير سوف يحمل رقم ذاكرة العرض
PAINTSTRUCT  ps;
RECT         rect;
static BOOL  fFirstPaint = TRUE;
```

الخطوة العاشرة والأخيرة

- في هذه الخطوة سوف نستعرض أولاً الأمر الشرطي (switch)

```
switch (iMsg)
{
case WM_KEYDOWN :
    switch (wParam)
    {
- في حالة ضغط المستعمل للمفتاح escape أو F12 يقوم البرنامج بالخروج:

        case VK_ESCAPE:
        case VK_F12:
            PostMessage (hwnd, WM_CLOSE, 0, 0);
            return 0;
    }
    return 0;
```

case WM_PAINT :

- الآن تقوم وظيفة BeginPaint بإعداد النافذة المطلوبة للرسم (بمعنى تقوم بتجديد جميع المكونات الظاهرة لتلك النافذة على الشاشة) ثم تقوم بعد ذلك بتعبئة المتغير PAINTSTRUCT (هذا المتغير من نوع Struct) بالمعلومات عن مكونات الرسم.

```
Hdc = BeginPaint (hwnd, &ps);
```

- بعد ذلك تقوم وظيفة GetClientRect بتعبئة المتغير rect بحجم مستطيل النافذة (النافذة مثل شاشة الكمبيوتر هي عبارة عن مستطيل ذي أبعاد معينة)

```
GetClientRect (hwnd, &rect);
```

- الآن نقوم بإظهار الجملة "Hello World" في منتصف الشاشة :

```
DrawText (hdc, "Hello World!!", -1, &rect, DT_SINGLELINE |  
DT_CENTER | DT_VCENTER);
```

- ثم بعد ذلك تقوم وظيفة EndPaint بإنهاء وظيفة الرسم على نافذة البرنامج ويقوم برنامجنا باستدعاء هذه الوظيفة في كل مرة ينتهي فيها من رسم (أي إظهار) شيء على الشاشة :

```
EndPaint (hwnd, &ps);  
if (fFirstPaint)
```

```

{
    ShowCursor (TRUE);
    fFirstPaint = FALSE;
}
return 0;

```

- وفي آخر هذه الخطوة نقوم باستدعاء وظيفة PostQuitMessage التي بدورها تقوم بإرسال الرسالة WM_QUIT إلى نظام التشغيل وذلك لإنهاء عمل البرنامج الذي بدوره يقوم بإنهاء الدورة اللانهائية لرسائل ويندوز في برنامجنا وإعطاء التحكم مرة أخرى إلى نظام التشغيل. وهذه هي الطريقة المثلى لإنهاء عمل برنامج ويندوز (طبعا هناك طرق أخرى لإنهاء عمل برامج ويندوز).

```

case WM_DESTROY :
    PostQuitMessage (0);
    return 0;
}

```

- وأخيرا في حالة وصول رسالة من ويندوز ليس لبرنامجنا علاقة بها (مثلا يرسل نظام التشغيل رسالة حركة الفأرة ونحن في هذا البرنامج لاتهمنا تلك الرسالة) فإن هذه الوظيفة تقوم بإعادتها إلى نظام التشغيل:

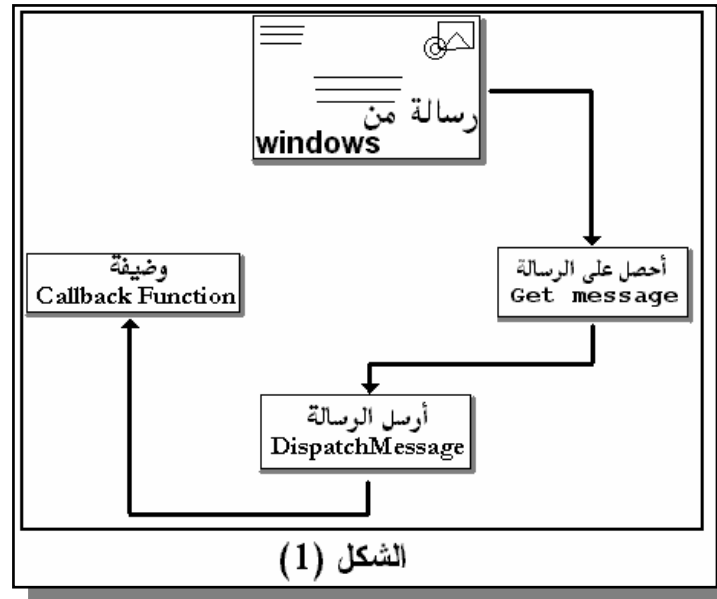
```

return DefWindowProc (hwnd, iMsg, wParam, lParam);
}

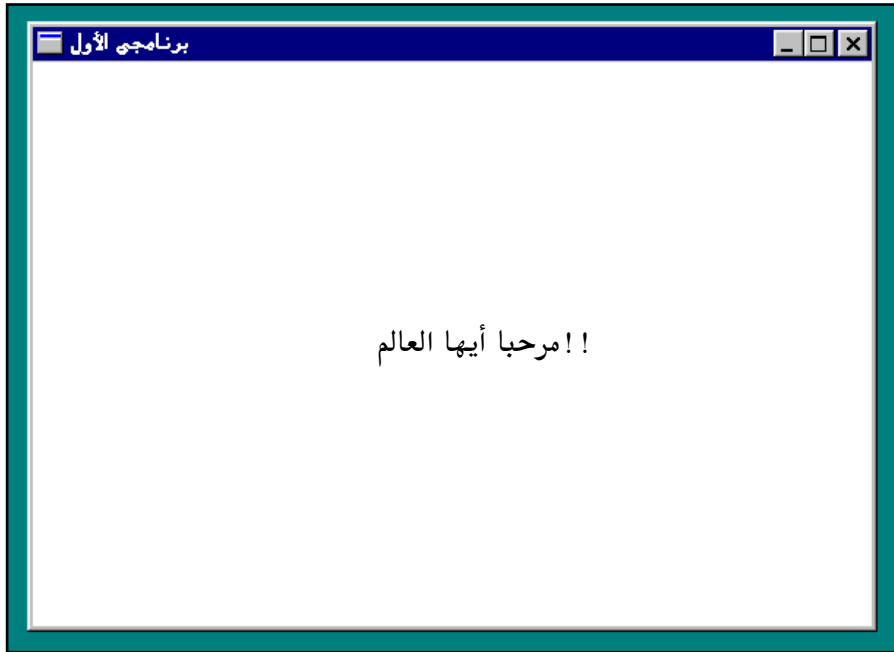
```

هذه نهاية وظيفة "WinProc"

هذا كل ما في الأمر ،ولو أنك درست جميع برامج نظام ويندوز فإنك سوف تلاحظ تشابهها مع هذا البرنامج. في هذه النقطة دعني أقول بأن البرنامج السابق هو مفتاح سر البرمجة على نظام ويندوز لذلك راجعة جيداً قبل أن تكمل قراءة هذا الكتاب. لا أعتقد أننا نحتاج أن نتعمق أكثر في برمجة Win32 لأن ذلك خارج نطاق موضعنا في هذا الكتاب وهناك الكثير من الكتب المختصة في برمجة النوافذ والتي تتكلم بشكل اعمق عن برمجة Win32. وكل ما يجب معرفته هو أن إضافة جميع عناصر DirectX سهلة عند معرفتك لهذا البرنامج. من هذه النقطة كل البرامج متشابهة (بالنسبة لقوانين البرمجة) لكل من النظامين Dos و Windows (البرنامج موجود في القرص المدمج في الملف win32).



ملاحظة: سألني الكثير من الأخوة عن طريق بريدي الإلكتروني عن كيفية البرمجة باللغة العربية، والمسألة في الحقيقة في غاية السهولة. مثلاً لتحويل البرنامج السابق إلى اللغة العربية سوف تحتاج طبعاً إلى نظام Windows العربي (النظام الجديد Windows2000 يحتوي على اللغة العربية حتى في الإصدارات الأجنبية) ، وعلى فرض ذلك كل ما تقوم به هو تحويل الكلمات الظاهرة على الشاشة من اللغة الإنجليزية إلى اللغة العربية. مثلاً في هذا الدرس كل ما قمت به لتحويل البرنامج إلى اللغة العربية هو كما يلي: في الخطوة الرابعة قمت بتحويل عنوان البرنامج من الكلمة الإنجليزية "Skelo 32" إلى الكلمة العربية "برنامجي الأول" وفي الخطوة العاشرة قمت بتحويل الكلمة الإنجليزية "Hello world" إلى العبارة العربية "مرحباً أيها العالم". فأصبح البرنامج الآن عربياً. نعم الأمر بهذه السهولة، أنظر إلى شكل البرنامج بعد التعديل:



- إذا كان كل ما يقوم به برنامج بهذا الحجم هو إظهار جملة على الشاشة ، إذاً فما هو حجم برنامج يقوم بأكثر من ذلك؟

الكثيرين من مبرمجي النظام Dos قد يسألون هذا السؤال ، فنستطيع أن نقوم ببرمجة برنامج مشابه لهذا على نظام Dos في أسطر معدودة لاختلاف ذلك النظام عن ويندوز. والإجابة على هذا السؤال بسيطة فبرنامجنا هنا لا يقوم بإظهار جملة على الشاشة فقط ولكن يقوم بإنشاء نافذة وإظهارها على الشاشة ثم يقوم بترشيح رسائل ويندوز و استخدامها عند الحاجة كما يقوم باحترام البرامج الأخرى والعاملة في نفس الوقت على نفس النظام. ويجب معرفة أننا نقوم بهذه العملية مرة واحدة فقط ثم بعد ذلك يكون حجم البرنامج هو نفسه على أي نظام(بسبب طبيعة البرمجة الكائنية باستخدام C++ فإن البرامج المكتوبة بها تكون "أصغر" بكثير عما هي عليه في الواقع).

هل أستطيع استخدام الشاشة كاملة "Full Screen" دون استخدام DirectDraw؟

نعم تستطيع وبكل سهولة كل ما عليك عمله هو بعض التغييرات في وظيفة CreateWindowEx في الخطوة الرابعة كما يلي:

- غير صفات النافذة إلى :

WS_POPUP | WS_VISIBLE,

- غير القيمة المبدئية للموقع x :

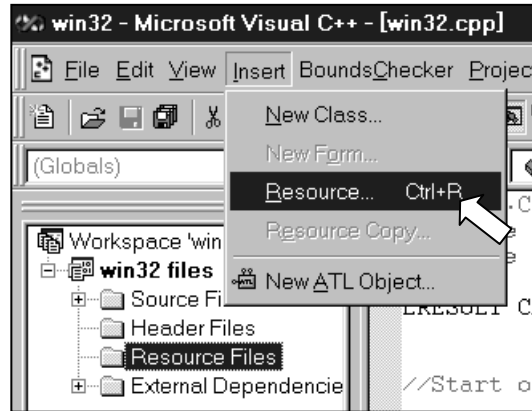
0,

- غير القيمة المبدئية للموقع y :
- 0,
- غير القيمة المبدئية لحجم x :
- GetSystemMetrics(SM_CXSCREEN),
- غير القيمة المبدئية لحجم y :
- GetSystemMetrics(SM_CYSCREEN),

طبعا البرنامج موجود كاملاً على القرص المدمج مع هذا الكتاب , وسوف نعتمد هذا البرنامج مع بقية برامج الكتاب. وسوف نقوم بالتغييرات المناسبة عند حاجتنا لها.

كيف تضيف Resources لبرنامج Win32

نستطيع استغلال مميزات برامج النوافذ عن طريق إضافة ما يسمى بـ Resources وفي هذا المثال ستري كيف نستطيع إضافة ايكونه (الأيكونه ما هي



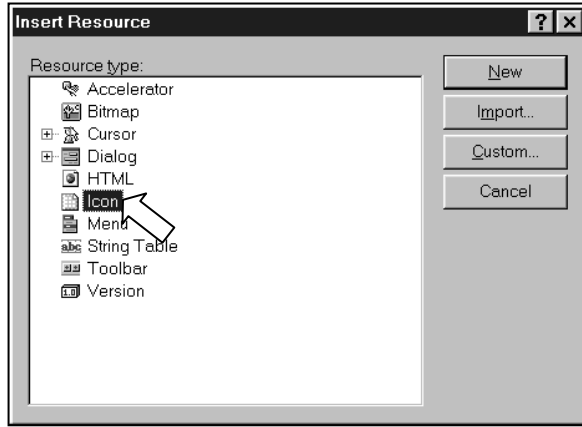
إلا صورة صغيرة تعبر بها ويندوز عن البرنامج مثل باقي البرامج الكثيرة في جهازك والتي لها ايكونات مختلفة) أولاً سنقوم بإضافة صورة الأيكونه إلى Resource

(وهو المكان الذي يحتفظ فيه البرنامج بالصور التي يستعملها أو أي ملفات

أخرى ويصبح جزء من البرنامج (exe) عن طريق اختيار **Insert** ثم **Resource** كما في الصورة.

ملاحظة : هذه الإضافات موجودة في الملف **Win32+** على القرص المدمج الملحق بالكتاب.

❖ بعد ذلك سوف ترى الشاشة التالية :



❖ نختار **Icon** "أيقونه" من القائمة. بعد ذلك نختار **New** من القائمة على اليسار إذا كنا لا نملكها وسنقوم برسمها. أو نختار **Import** إذا كانت موجودة على ملف سبق لنا وأنتجناه (هناك الكثير من البرامج المختصة في رسم الأيقونات).

في هذا المثال سوف نختار **New**.

❖ بعد ذلك تظهر لنا شاشة مشابهة لبرامج الرسم نقوم باستعمال الأدوات المتوفرة لرسم الأيقونة المناسبة لبرنامجنا.

❖ الآن نقوم بحفظ الأيكونه عن طريق اختيار الأمر **File** من القائمة العلوية ثم **Save As....** (الملف ينتهي بالحرفين **rc** وتستطيع تسميته ما تشاء ولكن لهذا المثال سوف نختار الاسم التلقائي له وهو **Script1.rc**).

ملاحظة : إذا كان الملف **Script1.rc** موجود سابقا فقم بحفظه مرة ثانية. بالإضافة إلى ذلك يقوم المصرف بإضافة الملف **resource.h** لنا , وسوف يحمل التعاريف المطلوبة للبرنامج لكي يتعرف على الملفات المختلفة (لأننا نستطيع إضافة أي نوع من الملفات تقريبا مثل ملفات صوتية أو فيديو أو صور هذا بالإضافة للقوائم إذا أردنا).

❖ بعد أن حفظنا الملف **Script1.rc** نقوم باختيار **Project** من القائمة العلوية ثم منها نختار الأمر **Add to Project** ثم نختار الأمر **Files...** سوف نرى بعد ذلك شاشة تسمح لنا باختيار الملفات التي نود إضافتها لمشروعنا , نختار الملف **Script1.rc** الذي حفظناه في الخطوة السابقة.

❖ في هذه المرحلة نكون قد انهينا الجزء الشكلي وعلينا أن نقوم ببعض التغييرات في الشفرة البرمجية لـ **Win32** حتى نستطيع رؤية الأيكونه الجديدة.

نقوم أولا في أعلى الملف بإضافة ما يلي :

```
#include "resource.h"
```

❖ بعد ذلك نذهب إلى الخطوة الثانية ونغير الأمرين التاليين :

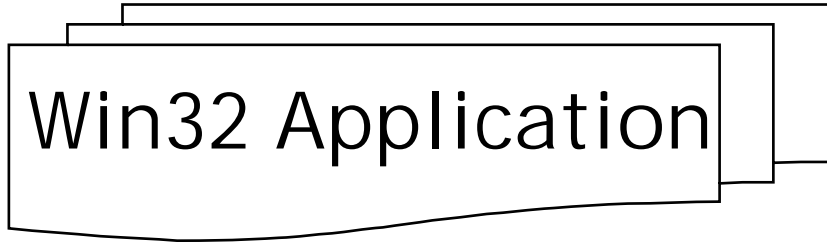
```
wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION);
wndclass.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
```

إلى ما يلي :

```
wndclass.hIcon = LoadIcon (hInstance, (LPCTSTR) IDI_ICON1);  
wndclass.hIconSm = LoadIcon (hInstance,(LPCTSTR) IDI_ICON1);
```

وكما ترى أن الشيء الجديد هنا هو المتغير `IDI_ICON1` وهو اسم الأيقونه ,
وتستطيع الحصول عليه من الملف `resource.h` الذي أنتجه المصرف لنا.

هذا كل ما نحتاج عملة لكي تضيف `Resources` لبرنامج `Win32` ولاحظ أن كل
هذه العملية سوف تتم بشكل تلقائي لنا عندما نستخدم `Win32 Application`.



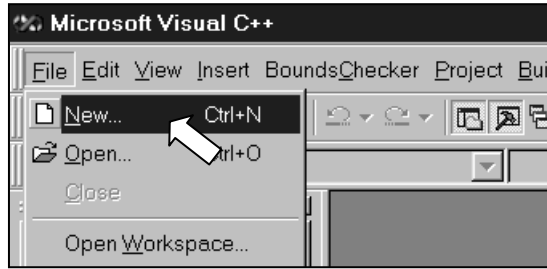
البرنامج السابق `win32` كما سبق وذكرنا هو الهيكل العظمي لكل البرامج التي
تعمل على أنظمة النوافذ ، ولهذا السبب علينا كتابة نفس الشفرة البرمجية كل ما
احتجنا لكتابة برنامج جديد. فقامت شركة `Microsoft` بإضافة ميزة جديدة (هي
ليست جديدة ولكنها أصبحت الآن أكثر تطورا) لبرنامج التطوير `Visual C++`
وتسمى بـ `Win32 Application` ويقوم بسؤالنا بعض الأسئلة البسيطة التي على
أساسها يقوم بكتابة الشفرة البرمجية لنا. طبعاً هذا لا يعني انه يستطيع كتابة

برامج كاملة ولكنها طريقة فعالة لبداية مشروع جديد , فيغنيينا بذلك عن الحاجة لإعادة كتاب نفس الشفرة البرمجية في كل مرة.
في هذا الفصل سوف نطلع علي كيفية استخدام **Win32 Application** مع برنامج التطوير **Visual C++** الإصدار السادس.

- نقوم أولاً بتشغيل برنامج التطوير **Visual C++** عن طريق اختيار من قائمة البرامج:

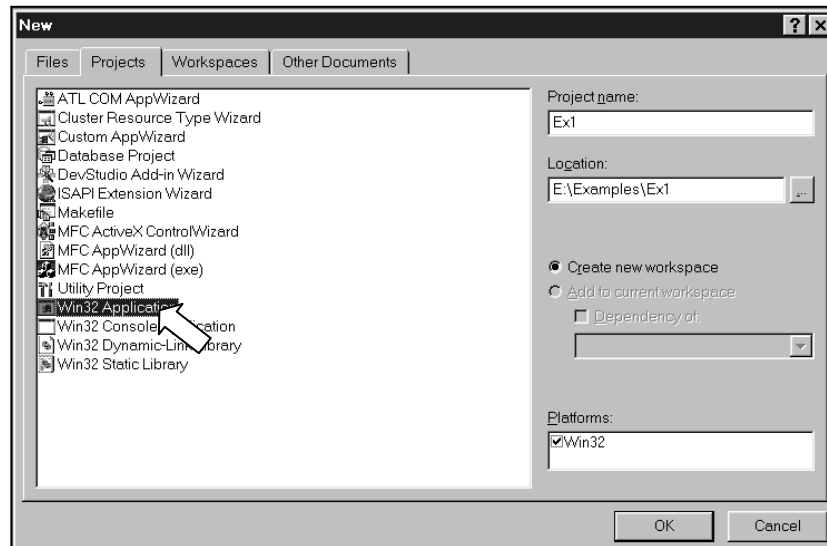


- بعد ذلك نختار **New** من الأمر **File** في القائمة العلوية :



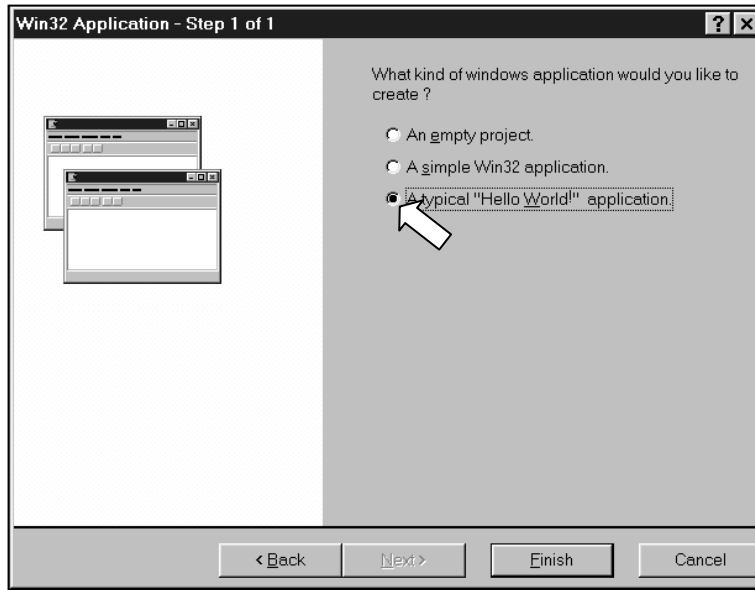
- سوف نرى نافذة جديدة (انظر في الصفحة التالية) نقوم من خلالها باختيار **.Win32 Application**

كذلك في المستطيل تحت الجملة **Project name** نقوم بكتابة اسم المشروع كما نستطيع اختيار أي اسم نريده. في هذا المثال اخترنا الاسم **Ex1**. في المستطيل اسفل ذلك نرى الجملة **Location** مكتوبة. في هذا المستطيل نقوم بكتابة المكان الذي ستحفظ فيه الملفات.



ثم نضغط على الزر **OK**

- بعد ذلك سوف نرى الشاشة التالية :



من خلال هذه الشاشة نقوم باختيار:

● A typical “Hello World” application

هذا يعني أننا نريد برنامج Win32 يقوم بعرض كلمة “Hello World” كما فعلنا في مثالنا.

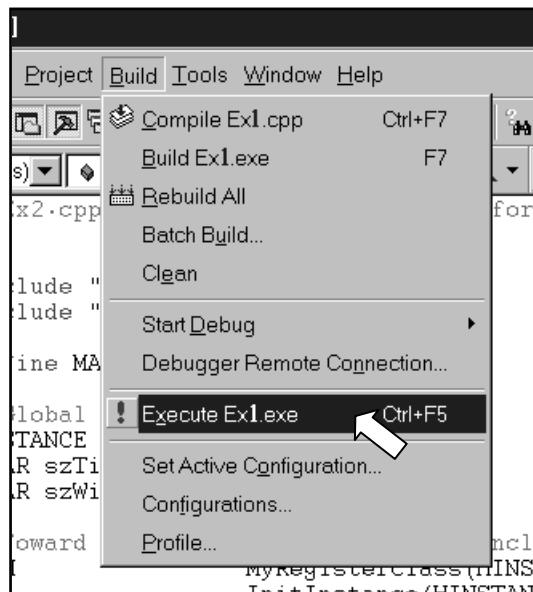
بعد ذلك نقوم بالضغط على الزر “Finish”.

ثم في الشاشة التي تلي ذلك نُسأل فيما إذا كُنَّا متأكدين من اختياراتنا ، و

نجاوب بنعم عن طريق الضغط على الزر “OK”.

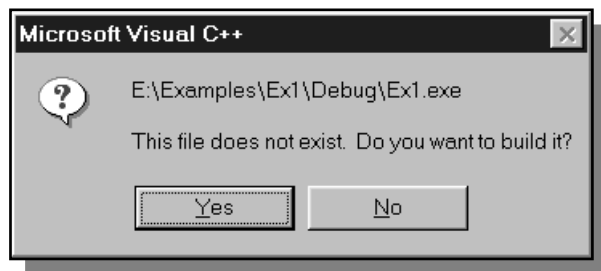
هذه كل الخطوات ، الآن لدينا شفرة برمجية كاملة لـ Win32.

- الآن سنقوم ببناء هذه الشفرة البرمجية لإنتاج برنامجنا :



سنقوم باختيار **Build** من القائمة العلوية لبيئة التطوير. من قائمة **Build** نختار الأمر **Execute Ex1.exe** والذي سيقوم ببناء وتشغيل البرنامج.


لأننا نبني الشفرة البرمجية لأول مرة يقوم المصدر بإعلامنا أن تشغيل البرنامج غير ممكن قبل بناء الملف التشغيلي **.exe**. ويقوم بسؤلنا ليبنيه قبل تشغيله , عن طريق الرسالة التالية :



طبعا نقوم بالضغط على زر الموافقة "Yes".

ثم تشغيل البرنامج عندها سنرى الشاشة الرئيسية :



وكما نرى فإن النتيجة هي نفسها التي حصلنا عليها من قبل ولكن مع إضافات بسيطة وهي أننا الآن لدينا قائمة علوية تحتوي على الأمرين **File** و **Help** كذلك سترى في أعلى يسار الشاشة الشكل  وهو عبارة عن ايكونه نعبر بها عن برنامجنا. مثلا لو نضرت إلى ملف التشغيل (exe) الناتج فسترى انه يظهر بنفس شكل الأيكونه.

وبما أننا في مثالنا الذي كتبناه في بداية هذا الفصل دون استخدام **Win32 Application** لم نقوم بهذه الإضافات فإن الشفرة البرمجية لها ليست موجودة.

وأخيرا نقول أنه ليس من المهم كيفية إنتاج برنامج **Win32** ولكن المهم أن تكون ملماً به. لأننا سوف نستعمله مع بقية أمثلة الكتاب. لذلك راجعة وتأكد من فهمه قبل أن تكمل. وهذا سوف يساعدك هذا على فهم جميع برامج النوافذ.

الفصل الرابع

Tiles

Tiles

الحاجة أم الاختراع ، هذا هي العبارة المناسبة لشرح أحد أجمل المواضيع في برمجة الصوت والصورة على الكمبيوتر وخاصة في برمجة الألعاب الإلكترونية. ما هي التايلز؟ لماذا نستخدمها؟ وما هي فائدتها؟ وكيف ننشئها؟ في الكتاب السابق "برمجة ألعاب الكمبيوتر على النظام windows95" ذكرنا هذا الموضوع واستخدمنا مثال لذلك ولكننا لم نتطرق بعمق للإجابة عن الأسئلة الكثيرة لهذا الموضوع. وعند ذكر التايلز فإننا لا نرى بدأً من التكلم عن إنشاء صفحات الإنترنت فهي كذلك تستخدم التايلز كخلفية لها ، وإذا كان لديك عزيزي القارئ بعض المعرفة في إنشاء صفحات الإنترنت فسوف تتعرف مباشرة على هذا الموضوع الشيق.

بعض الإحصائيات :

(ملاحظة : 1 ميغا بايت = 1024 كيلو بايت ، 1 كيلو بايت = 1024 بايت)
 في برمجة ألعاب الكمبيوتر بشكل خاص وبرمجة الصوت والصورة بشكل عام نرى الحاجة الماسة إلى استخدام خلفيات متحركة. في حالة لو أننا استخدمنا البعد 640x480 للشاشة (يطلق عليها درجة صفاء الشاشة وتعبر هذه الأرقام عن عدد النقاط الموجودة على الشاشة في وقت واحد) وأردنا استخدام خلفية تغطي هذه الشاشة بكاملها فإننا سوف نحتاج إلى صورة بنفس البعد لكي تقوم بذلك. ولكن صورة بهذا البعد (640x480) سوف تأخذ من الذاكرة 308280 بايت أو 301 كيلو بايت (هذا يعني أن كل نقطة على الشاشة سوف تأخذ بايت واحد. يوجد هنا

480 نقطة ضرب 640 نقطة ويساوي 307200 نقطة هذا بالإضافة إلى تعاريف الصورة والتي تأخذ حيز 1080 بايت إضافي) كما في المعادلة التالية:

$$301 \text{ كيلو بايت} = 1080 + 640 \times 480 = 308280 \text{ بايت}$$

وهو حجم الصورة في الذاكرة ، هذا طبعا على أساس أن الصورة تحتوي على 256 لون أو تعرف بـ 8 Bit وفي حالة لو أردنا استخدام صور تحتوي على 65536 لون (تعرف بـ 16 Bit) فإن حجم الصورة في الذاكرة سوف يتضاعف ثلاث مرات (لأن كل نقطة سوف تأخذ ثلاثة بايت لتخزين الثلاث ألوان الرئيسية: الأحمر، الأخضر والأزرق) فيصبح الحجم كما في المعادلة التالية:

$$900 \text{ كيلو بايت أو } 921658 = 58 + 3 \times 640 \times 480 \text{ بايت تقريبا}$$

لاحظ أن تعاريف الصورة تأخذ فقط 58 بايت في حالة الـ 16 Bit بدل 1080 بايت في حالة الـ 8 Bit وذلك لأن كل نقطة تحتوي على كل المعلومات المطلوبة لمعرفة لونها أما في حالة استخدام 256 لون فإن هناك جدول (يعرف بلوحة الألوان "Palette") في بداية ملف الصورة يقوم بتخزين رقم للون كل نقطة في الصورة وبالتالي يأخذ حيز أكبر.



الهدف من كل الإحصائيات السابقة هو إعطائك فكرة عن حجم الصور المستخدمة في الذاكرة. ونحن نعلم بأن المستخدم قد لا يحتوي حاسبة الآلي على أكثر من 2 ميجا بايت (أي 2048 كيلو بايت) من ذاكرة الفيديو (وهي الذاكرة الموجودة على بطاقة العرض) وبالتالي استخدام صورتين فقط من الحجم السابق (الـ 16 Bit مثلا) سوف يأخذ تقريبا كل حيز ذاكرة الفيديو وبالتالي سوف تقوم DirectDraw

باستخدام ذاكرة النظام. وطبعاً هذا ليس أمر جيد لسرعة برنامجنا (راجع موضوع "الذاكرة" لتعرف السبب).

1

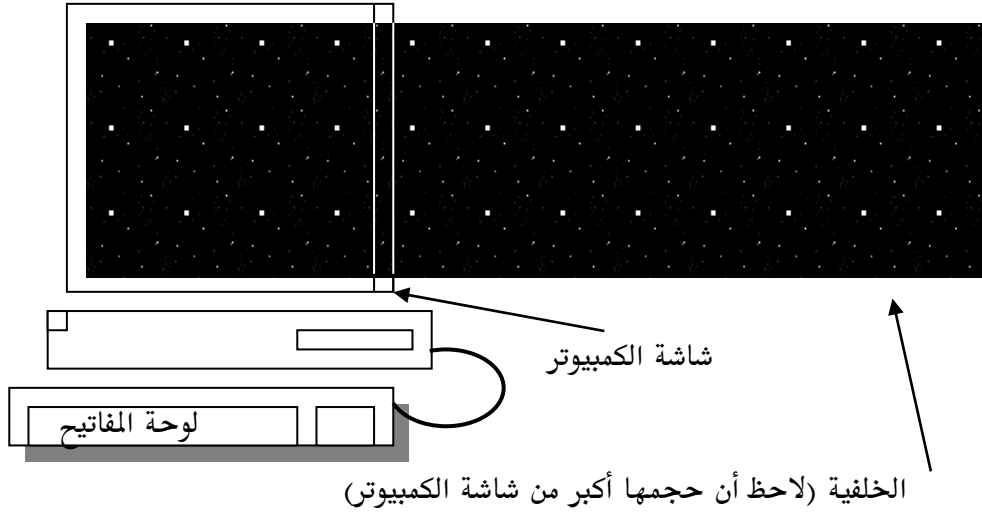
المعضلة أننا نريد أن نستخدم خلفيات متحركة، وبالتالي سوف تكون تلك الخلفيات أكبر من حجم الشاشة (الذي اتفقنا أن يكون 640x480) بكثير. ولو استخدمنا صور بحجم الشاشة لنعبر عن الخلفية المتحركة فإننا سوف نواجه مشكلة حجم ذاكرة الفيديو (يمكننا استخدام صور بحجم الشاشة إذا كان كل ما نحتاجه هو صورتين أو ثلاث مثلاً من نوع الـ 8 Bit).

ما هو الحل إذاً ؟

نعم هذا صحيح ، استخدام التايلز هو الحل ، ولكن لماذا ؟ ولإجابة هذا السؤال يجب أولاً أن نعرف ما هي التايلز. التايلز هي جمع كلمة تايل وهي صورة مربعة صغيرة الحجم (رخامه أو بلاطة) بحيث يمكن إعادة استخدامها كلما أردنا في الخلفية. أنظر مثلاً إلى صفحات الإنترنت فهي تستخدم صورة صغيرة مكررة كخلفية بحيث تملأ الشاشة وبما أنها صورة صغيرة فهي لا تأخذ الكثير من الذاكرة واستخدامها يعطينا القدرة على تكرارها حسب مشيئتنا.

لنرى صورة التايل التالية على سبيل المثال:

هذه مثلا صورة صغيرة للفضاء ، لو لاحظت جيدا سترى أننا عند تكرار هذه الصورة نستطيع أن نجعل الخلفية بالحجم الذي نريد كما في الشكل التالي :



وهكذا نستطيع أن نكرر الصورة كما نريد حتى نصل إلى الحجم المطلوب. وقد يسأل القارئ الكريم لماذا لا نستخدم صورة واحدة للخلفية بدل تكرار تلك الصورة الصغيرة مرات عديدة؟ السبب كما شرحنا سابقا هو "الذاكرة" فهذه الصورة الصغيرة تأخذ حجم من الذاكرة لا يتعدى 3 كيلو بايت فقط.

الملاحظ في المثال السابق بأن صورة الفضاء ينقصها شيء من الواقعية وذلك لأننا نرى التكرار بشكل واضح وحل هذا المشكلة بسيط للغاية فكل ما ينبغي فعله هو استخدام أكثر من تايل لإنشاء المشهد الخلفي وبهذا نستطيع أن نعطي منظر خلفي مقنع.

كيف ننشئ التايلز ؟

التايلز صور مثل أي صور أخرى ، بمعنى أننا نحتاج إلى برنامج إنشاء وتعديل الصور (مثل البرنامج PhotoShop أو PaintPro) لنستطيع إنشاء هذه النوعية من الصور. كذلك نحتاج إلى برنامج آخر نقوم من خلاله بإنشاء خريطة للخلفية في برنامجنا.



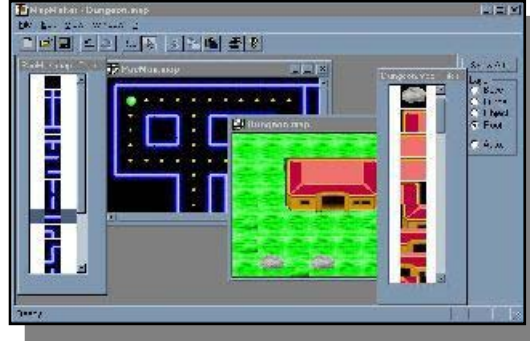
ماذا تعني ببرنامج نقوم من خلاله بإنشاء خريطة للخلفية في برنامجنا؟؟

بعد إنشاء صور التايلز المطلوبة فإننا سوف نحتاج لطريقة (أو في الحقيقة لبرنامج) نقوم من خلاله بإنشاء خريطة لشكل الخلفية المطلوبة ، بمعنى أننا نريد أن نرى شكل الخلفية الناتجة عن صور التايلز ؟ بالنسبة للأخوة القراء الذين سبق لهم استخدام مكتبة الألعاب (Game.lib وهي المكتبة المستخدمة مع كتابي السابق) فإن الطريقة كانت إنشاء ملف من نوع txt نضع فيه أماكن كل صورة تايل. ولكن في الحقيقة هذه الطريقة متعبة شيئا ما ، كما أنها ليست مجدية عند إنشاء خلفية كبيرة الحجم ، لذلك في هذا الكتاب سوف نقوم باستخدام برنامج يطلق عليه "MapMaker" يقوم بإنشاء الخلفيات المطلوبة وبشكل دقيق (طبعا البرنامج موجود من ضمن القرص المدمج الملحق بالكتاب). فكل ما علينا فعله هو إنشاء صور التايلز المطلوب استخدامها كخلفيات ثم تعبئتها في هذا البرنامج بحيث نقوم بعد ذلك بترتيب هذه الصور حسب مشيئتنا لإنشاء الخلفية المطلوبة (سنقوم بشرح تفصيلي لهذه العملية).

برنامج MapMaker

أحد البرامج الموجودة على الإنترنت ، قام بتصميمه المهندس Lennart.Steinke وتستطيع أن تذهب إلى صفحته في الإنترنت على العنوان التالي:
<http://www.geocities.com/SiliconValley/Vista/6774/>

وأنا شاكر للمهندس لينارت كل المساعدة الشخصية التي قدمها لي في كيفية استخدام هذا البرنامج كذلك في كتابة الكائن (Class Object) لهذا الكتاب والذي من خلاله نستطيع استغلال مميزات هذا البرنامج في برامجنا التي سوف



نستخدمها في هذا الكتاب.

برنامج MaapMaker

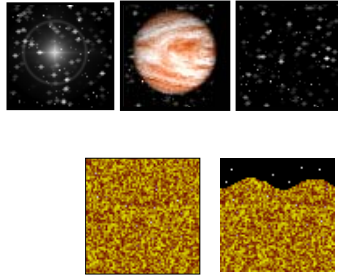
ويوزع بالمجان (يوجد على القرص المدمج الملحق بالكتاب) وسوف نقوم باستخدامه في تصميم خلفيات برامجنا في هذا الكتاب. (حاليا أقوم باستخدام هذا البرنامج في تصميم خلفيات لعبة الكمبيوتر "فضاء" والموجودة أيضا على القرص المدمج). يوجد

في هذا البرنامج إمكانيات كثيرة ولكننا سنركز على أهم الأوامر التي عن طريقها سنقوم بتصميم الخلفيات المطلوبة.

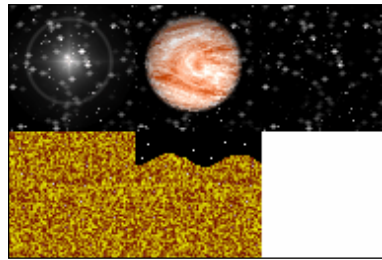
• Autograb.exe

قبل البدء في استخدام هذا البرنامج طبعاً يجب أولاً أن نكون قد قمنا بتصميم صور التايلز المطلوب استخدامها وتخزينها على ملف من نوع "bmp" أو "Pcx". ثم نقوم بتحويلها إلى نوعية "tlf" وهي النوعية الخاصة لهذا البرنامج وتتم هذه العملية عن طريق استخدام برنامج آخر (أيضاً من ضمن القرص المدمج الملحق بالكتاب) يطلق عليه "Autograb.exe" يقوم بهذه العملية.

لنفرض مثلاً كخطوة أولى لدينا صور التايلز التالية:



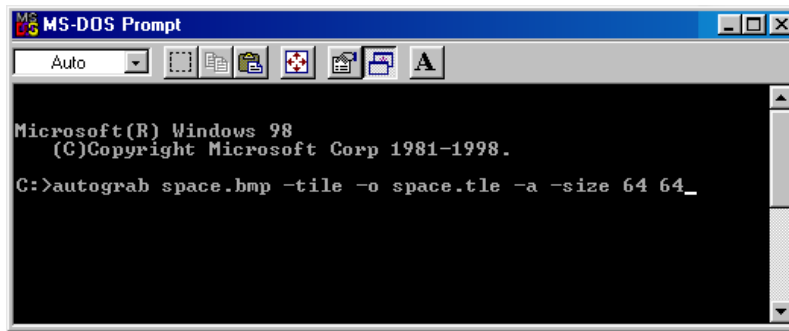
في هذه الحالة نحن نعلم حجم كل صورة تايل، هنا مثلاً استخدمنا 64x64. ونستطيع استخدام أي برنامج لتعديل الصور (مثل PhotoShop) بحيث ننشئ صورة جديدة تضم جميع صور التايل التي نحتاجها. في هذا المثال قمنا برسم ما يلي كخطوة ثانية:



لاحظ أن كل ما قمنا به هو وضع صور التايلز في صورة جديدة لنطلق عليها مثلا الاسم "Space.bmp" ونحن نعلم حجم كل صورة تايل، الآن في الخطوة الثالثة نقوم باستخدام الأمر "Autograb.exe" لنحصل على صورة من نوعية البرنامج "MapMaker" كما يلي :

ملاحظة : لكي نشغل أي أمر من C:> على أنظمة النوافذ نختار البرامج من قائمة البداية ثم نختار MS-DOS Prompt.

```
c:> autograb space.bmp -tile -o space.tle -a -size 64 64
```



بعد كل شرطة "-" يوجد اختيار ، مثلا في الحالة السابقة استخدمنا الخيارات التالية :

-tile ويعني أننا نريد ملف جديد من النوع tle (لاستخدامه مع برنامج "MapMaker")

-o ويعني أننا نريد الملف الخارج أن يحمل اسم معين (في هذه الحالة space.tle)

a- ويعني أننا نريد من برنامج autograb أن يقسم الصورة إلى صور تايلز ذات حجم معين سوف يكون إدخاله عن طريق الأمر -size- ويجب استخدام هذين الأمرين معا.

-size- يعني أننا سنقوم بإعطاء حجم كل صورة تايل ويجب أن يستخدم مع الأمر السابق.

64 x64 هو حجم صورة التايل.

هناك خيارات أخرى لهذا الأمر وتستطيع معرفتها عن طريق كتابة:

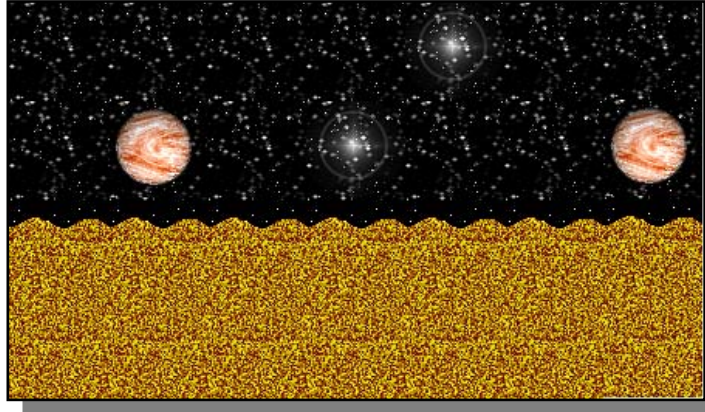
c:> autograb/?

بعد أن نحصل على الملف المطلوب ، في هذه الحالة سيكون :

space.tle

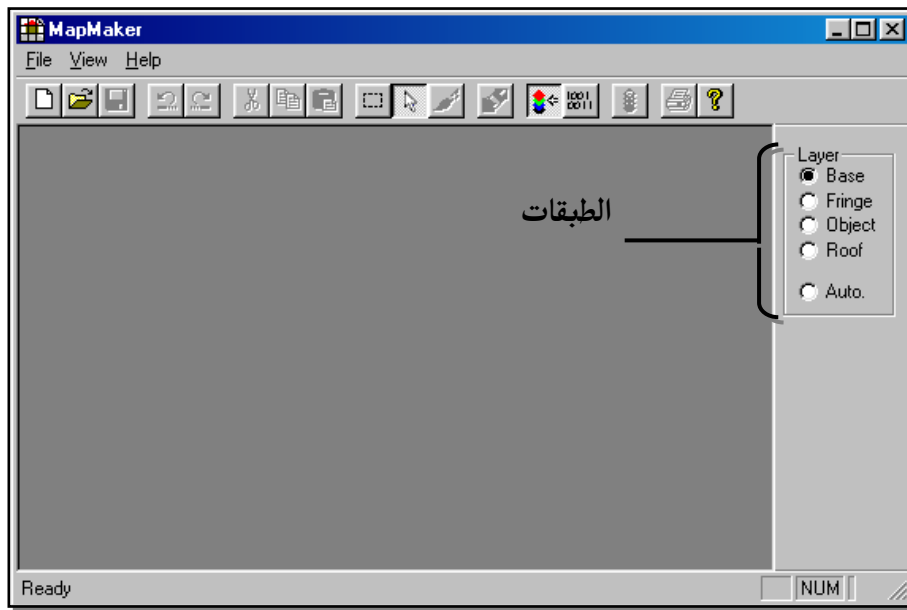
نقوم بتشغيل برنامج "MapMaker" وإنشاء ملف جديد لخريطة الخلفية ونقوم بتعبئة ملف space.tle المحتوي لصور التايلز (سوف نقوم بشرح هذه العملية بشكل مفصل بعد قليل) .

نستطيع بعد استخدام البرنامج "MapMaker" على الخلفية المطلوبة وبأي حجم :



طريقة إنشاء خلفية متحركة لصور التايلز باستخدام برنامج MapMaker

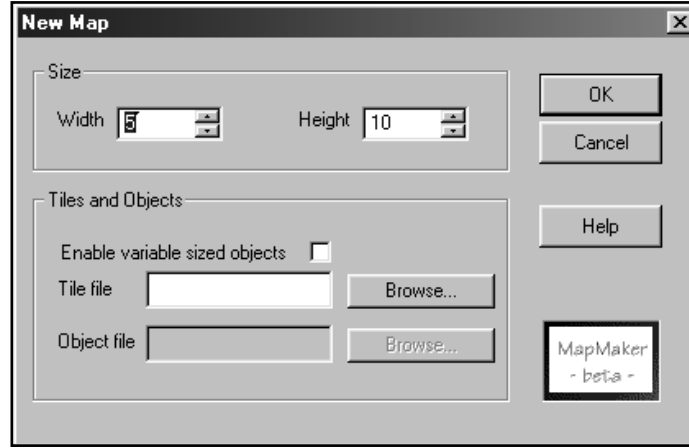
في الخطوة الأولى سوف نحتاج إلى عدد من صور التايلز ونحولها كما فعلنا سابقاً إلى نوعية الملفات التي يستطيع البرنامج MapMaker التعرف عليها. الآن وبعد أن أصبحنا نملك صورة التايلز التي نود استخدامها على هيئة ملف tile نستطيع أن نبدأ تشغيل برنامج MapMaker ونراه في الشكل التالي :



الشاشة الرئيسية

وكما نرى أنها بسيطة ولكنها تحتوي على جميع الإمكانيات التي سنحتاجها وسوف نتعرف على كل أمر حسب الحاجة.

في الخطوة الثانية وبعد أن نرى الشاشة السابقة نقوم باختيار الأمر New من قائمة File عندها سوف نرى الشاشة التالية (انظر أعلى الصفحة القادمة) :

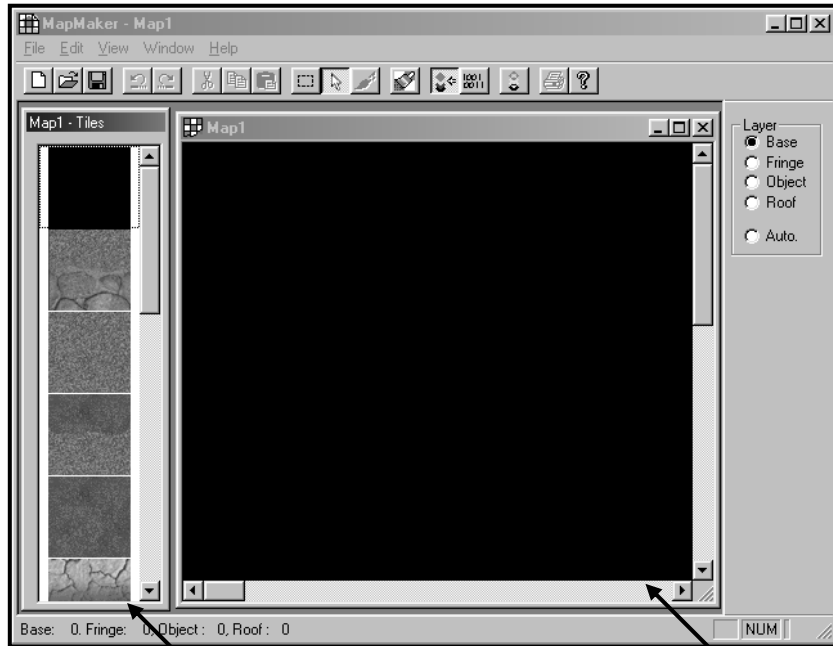


وكما نرى هنا أن البرنامج يطلب منا حجم "Size" الخريطة الخلفية وذلك بتحديد عدد الصور التايلز التي تمثل عرض "Width" الخريطة وعدد صور التايلز التي تمثل ارتفاع "Height" الخريطة عن طريق وضع الأرقام المناسبة. لهذا المثال سنستخدم الرقم 13 للعرض والرقم 70 للارتفاع.

بعد أن حددنا حجم الخريطة نقوم باستخدام زر البحث "Browse" لإدخال ملف صور التايلز tile والذي أنشأناه في الخطوة الأولى. (لهذا المثال سنقوم باستخدام ملف tiles.tile لبرنامج المثال الثامن أحد الأمثلة التي سنراها لاحقا لمكتبة AGDX بحيث سنرى هنا كيفية إنشائه ثم نرى لاحقا كيفية استخدامه في برنامج المثال الثامن).

يجب أن لا ننسى أن الهدف الرئيسي من كل هذه العملية هي الحصول على خريطة "Map" لصور التايلز واستخدامها في برامجنا. وهذه الخريطة هي عبارة عن ملف ينتهي بـ map ومن خلاله يتعرف الكمبيوتر على مكان كل صورة تايلز في الخلفية وكذلك يعطينا القدرة على إضافة بعض المميزات الخاصة لكل صور من صور التايلز إذا أردنا ذلك. وكذلك نستطيع إنشاء أكثر من طبقة لإضافة نوع من البعد الثالث للخلفية المتحركة وأشياء أخرى كثيرة يوفرها لنا هذا البرنامج.

الخطوة الثالثة وبعد تعبئة الملف tiles.tle ووضع الأرقام المناسبة للعرض والارتفاع نضغط على زر الموافقة "OK". بعدها سنعود إلى الشاشة الرئيسية للبرنامج ولكن هذه المرة تحتوي بداخلها على نافذتين إضافيتين :



نافذة صور التايلز

نافذة الخريطة

وكما تلاحظ أننا الآن نملك نافذة الخريطة والتي من خلالها سنقوم بوضع صور التايلز التي نختارها من نافذة صور التايلز.

سبق وأشرنا إلى أننا باستطاعتنا استخدام أكثر من طبقة للخلفية واستخدام أكثر من طبقة يضيفي شيء من الواقعية والبعد الثالث لخلفيات برامجنا كما يمكننا مثلاً في لعبة مغامرات أن نرى المنازل من الأعلى ثم حين دخول الممثل إلى منزل معين فإن السقف ينفتح بحيث نستطيع أن نرى ما داخل المنزل عن طريق التحكم في الطبقة التي تمثل سقف المنزل. ويحتوي برنامج MapMaker على أربع طبقات (Base , Fringe , Object , Roof) وهذه المسميات لا تعني شيء بالضرورة إنما مجرد دلالات للطبقة الأولى (الرئيسية) والثانية والثالثة والرابعة.

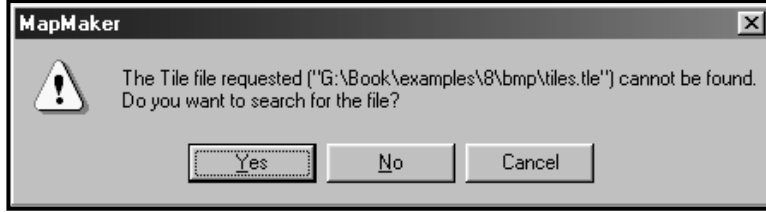
ويقوم هذا البرنامج بوضعك في الطبقة الرئيسية بشكل تلقائي ونحن في هذا المثال بقصد التبسيط سنعمل مع الطبقة الرئيسية فقط وإضافة أي طبقات أخرى هو ببساطة الضغط بإشارة الفأرة على أي منها ولكن إذا كنت تتعرف على هذا البرنامج للمرة الأولى أنصحك بعدم تجربة ذلك حتى ترى نتيجة هذا المثال أولاً.

الخطوة الرابعة نقوم بإنشاء الخلفية عن طريق اختيار صور التايلز المناسبة وهذه الخطوة بسيطة وممتعة جداً لأن كل ما نقوم به هو وضع صور التايلز في المكان المناسب ونملاً الخريطة.

الخطوة الخامسة والأخيرة هي أن نخزن ملف الخريطة على القرص الصلب , في هذا المثال سوف نطلق عليه اسم land.map . الآن نحن نملك ملفين ملف صور التايلز الذي أنشأناه في الخطوة الأولى وملف الخريطة الذي خزناه في الخطوة

الخامسة وهذين الملفين هما كل ما نحتاج لاستخدام هذه الخلفية في برامجنا فكل ما عليك فعله هو وضعهما في المكان المناسب لبرنامجنا لاستخدامهما. انظر إلى المثال الثامن من أمثلة AGDX لترى النتيجة بنفسك.

ملاحظة أخيرة : أحيانا كثيرة سوف تحتاج لتعبئة ملف الخريطة مرة أخرى إلى برنامج MapMaker وذلك لإضافة بعض التعديلات عليه. ستذهب إلى قائمة الملفات في الشاشة الرئيسية وتفتح ملف الخريطة عندها قد لا يجد برنامج MapMaker ملف التايلز التابع لتلك الخريطة وترى الرسالة التالية :



والتي تبغك بعدم قدرة البرنامج على إيجاد ملف التايلز المطلوب للخريطة وتساءلك إذا كنت تريد أن تبحث عنه بنفسك. طبعاً تجيب بنعم "Yes" وسوف ترى نافذة جديدة تسمح لك بالبحث عن الملف. اختار الملف المناسب وابدأ في التعديل.

الفصل الخامس

المكتبة

AGDX

المكتبة AGDX

ما هي مكتبة AGDX ؟ هي مكتبة تُستخدم لإنتاج برامج الصوت والصورة والألعاب والتي بدورها تستغل تكنولوجيا دايركت أكس. في الكتاب السابق "برمجة ألعاب الكمبيوتر" قمنا باستخدام مكتبة خاصة للألعاب يطلق عليها **GameLib** وهناك تشابه بين المكتبتين فكلاهما يمكن استخدامهما لإنتاج ألعاب الكمبيوتر ولكن تتميز مكتبة AGDX والتي سوف نستخدمها في هذا الكتاب عن **GameLib** بمميزات سوف نذكرها بعد قليل.

قصة المكتبة AGDX و CDX ؟

عند بداية تأليف هذا الكتاب كنت أقصد إرفاق المكتبة CDX (www.maidex.demon.co.uk) بعد موافقة صاحبها (Danny Farley) على أن تكون من ضمن مادة هذا الكتاب. والمكتبة CDX كانت من أول المكتبات المجانية التي تسهل عملية التعامل مع تكنولوجيا **DirectX**. وبطريقة تراكمية بعيدةً شيئاً ما عن الطريقة الكائنية وقريبة من برمجة النظام **Dos**. ولكن مع طول الوقت الذي أخذه هذا الكتاب ، اضطررت لترقية المكتبة CDX كثيراً عن شكلها الأصلي إلى الشكل الذي سترونه هنا وتغييرها إلى مكتبة خاصة بهذا الكتاب. وتم ذلك باستغلال آخر إصدارات هذه التكنولوجيا. وترقية الكثير من المميزات. ولكن بقي التوافق كما هو. بمعنى أن جميع البرامج التي تعمل مع الإصدار 1.5 للمكتبة CDX سيعمل مع مكتبتنا AGDX. وقد أرفقت جميع الأمثلة مع تحويلها إلى مكتبتنا والتي ستعطيك الصفات الرئيسية لكلا المكتبتين في القرص المدمج التابع للكتاب (تحت الملف `Extra\cdx2agdx`).

حاليا المكتبة CDX لم تعد قيد التطوير من قبل المبرمج الأصلي ولكن من قبل أشخاص آخرين قاموا بتغيير وتعديل الكثير من مميزات الأصلية. فوصلت لدرجة فيها الكثير من الاختلافات وعدم التوافق مع المكتبة AGDX.

طبعا الشفرة البرمجية كاملة للمكتبة AGDX مرفقة مع الكتاب لمن أراد تطوير المكتبة لمستلزماته الخاصة. كما أن هناك شرحا في الجزء الثاني من الكتاب عن كيفية التعامل مع تكنولوجيا DirectX ومع التركيبة الداخلية لهذه المكتبة.

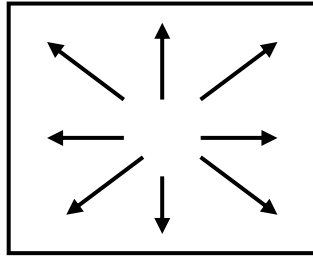
المميزات

- يمكن استخدامها لكل برامج الصوت الصورة سواء كانت برامج ألعاب أم برامج تعليمية أم برامج علمية إلى آخر ذلك من الأنواع المختلفة.
- تتميز عن المكتبة السابقة في أنها صممت بحيث نستطيع إضافة الأبعاد الثلاثية لها واستخدامها لإنتاج البرامج التي تعمل على أساس شبكات الاتصال.
- كما أننا نستطيع من خلالها استغلال الشاشة بكاملها أو استغلال نافذة من نوافذ ويندوز.
- هذه المكتبة تستخدم الوظائف وليس الكائنات في تعاملنا معها مما يجعلها أكثر سلاسة لمن تعود البرمجة على نظام Dos . (التركيب الداخلي لهذه المكتبة يعتمد بشكل رئيسي على الكائنات ولكن لا يهمننا هذا في الجزء الأول من هذا الكتاب. ونقوم بشرح تفصيلي لذلك في الجزء الثاني)

- مكتبة AGDX مجانية مع هذا الكتاب ويمكن تغييرها حسب احتياجك.

مكتبة AGDX مثل مكتبات DirectX توزع بالمجان وتستطيع أن تستخدمها حسب إرادتك بشكل مجاني وكما تستطيع إضافة أي تغييرات عليها إذا رغبت في ذلك (في الجزء الثاني من هذا الكتاب نقوم بشرح هذا الموضوع) ولكن لا تنخدع بكونها مكتبة مجانية. فهذه المكتبة من أجمل المكتبات المتوفرة لبرمجة تكنولوجيا DirectX وفي اعتباري هي أفضل بكثير من المكتبات التي تباع بمبالغ باهظة وتقوم بنفس العملية. والكثيرين يستخدمون هذه المكتبة في إنتاج ألعاب وبرامج صوت وصورة تستغل طاقة تكنولوجيا DirectX.

- خلفيات متحركة لثمانية جهات



ما هي الألعاب بدون خلفيات متحركة وما هي كل برامج الصوت والصورة بدون أن نرى تغيير مستمر في شكل الشاشة. الألعاب الشهيرة مثل "ماريو برذرز من شركة NINTENDO" و "سونك من شركة Siga" وتقريبا كل ألعاب

الحركة التي ظهرت إلى الآن تستخدم الخلفيات المتحركة. انظر في شرح الأمثلة كيفية سهولة استخدامها مع مكتبة AGDX.

- تحكم كامل في الممثلين "Sprite"

قد تحتاج في برنامج تعليمي إلى تحريك المدرس أمام السبورة وقد تحتاج في برنامج علمي تحريك الإلكترونات في السلك المعدني وكذلك قد تحتاج في لعبة من ألعاب الكمبيوتر تحريك الدبابات والطائرات على شاشة الكمبيوتر. نعم نحن بدون شك

نحتاج إلى الحركة على الشاشة ويتم ذلك عن طريق استخدام الممثلين وتعطينا هذه المكتبة التحكم الكامل في ذلك.

- تتعامل مع أي حجم للخلفيات المتحركة

الخلفيات المتحركة قد تكون بأي حجم أنظر إلى بعض ألعاب الفيديو ، سوف تلاحظ بأن الخلفية قد تستمر أحيانا طوال المرحلة. ويمكننا أن نقوم بنفس العملية هنا.

- خلفيات متحركة ملتفة ومنتهية

نستطيع كذلك أن نجعل الخلفية تلتف بحيث عند الوصول إلى آخرها ترجع مرة أخرى من البداية أو منتهية بحيث تتوقف عن نهايتها. كل هذا حسب رغبتنا.

- استخدام العنصر DirectInput للتعامل مع لوحة المفاتيح والفأرة وعصا الألعاب

العنصر DirectInput أحد عناصر تكنولوجيا DirectX نستطيع استغلاله بشكل كامل لإعطائنا التحكم المطلوب في البرامج التي سوف نكتبها.

- استخدام العنصر DirectSound لتعبئة واستخدام ملفات WAV الصوتية

العنصر DirectSound أحد عناصر تكنولوجيا DirectX نستطيع استخدامه للتحكم في سماع الأصوات في برامجنا (نستطيع إعطاء إحاء بالأبعاد الثلاثية للصوت) كما نستطيع التحكم في سرعة الصوت واتجاهه.

- تعبئة واستخدام ملفات MIDI الموسيقية.

ملفات MIDI المشهورة بصغر حجمها ونقاء الصوت (يختلف صوت كل ملف من جهاز لآخر اعتمادا على نوعية البطاقة المستخدمة ، ولكنها ذات مستوى جيد إلى ممتاز بشكل عام).

- استخدام وظائف الرسوم البسيطة مثل PutPixel (ضع نقطة) ، Line (خط) و Circle (دائرة) ... الخ
أحيانا كثيرة نحتاج إلى القيام ببعض الرسوم البسيطة كإضافة دائرة أو مربع لبرنامجنا. نستطيع أن نقوم بذلك بسهولة.
- استخدام وظائف التحكم في لوحة الألوان " Palette " مثل FadeOut (تحويل ألوان الشاشة إلى اللون الأسود) ، FadeTo (تحويل ألوان الشاشة إلى لون معين) و GreyScale (تحويل ألوان الشاشة إلى الأبيض والأسود)
عندما نتعامل مع لوحة ألوان تتكون من 256 لون نستطيع أن نستغل هذه اللوحة للقيام ببعض الخدع الجميلة على الشاشة. مثل FadeOut حيث تقوم هذه الوظيفة بتحويل ألوان الشاشة إلى اللون الأسود بالتدرج كما يحدث في الأفلام السينمائية وخدع أخرى سوف نتعرف عليها بالتفصيل لاحقاً.
- كتبت بلغة السي المحسنة C++ العامة وذلك لضمان عملها على أي برنامج لهذه اللغة يتعامل مع تكنولوجيا DirectX (مثل Visual C++ و Borland C++)
مع العلم أننا سوف نستخدم برنامج Visual C++ لإنتاج برامجنا في هذا الكتاب (وبالتالي طبعا انصح بشدة على اقتناء هذا البرنامج) إلا أن ذلك لا يعني أننا لا نستطيع استخدام برامج أخرى للغة السي مثل Borland.

كيف نستخدم المكتبة AGDX في برامجنا

مكتبة AGDX كتبت باستخدام لغة السي المحسنة "C++" بدون الاعتماد على مميزات لبرنامج معين وذلك حتى يمكن استعمالها مع اكبر عدد ممكن من برامج السي المحسنة "C++". وقد تم استخدام البرمجة الكائنية في تصميمها ، بمعنى أن كل مميزات المكتبة التي أطلعنا عليها نقوم باستغلالها عن طريق إنشاء الكائن المطلوب ثم استخدامه في برامجنا. وقد تتصور عزيز القارئ بأن هذا شيء معقد من أول وهلة ولكن لو أنك نظرت إلى الأمثلة سوف تكتشف سهولة هذه العملية فكل ما نقوم به هو كتابة عدة اسطر بسيطة نقوم عن طريقها بإنشاء الكائن المطلوب استخدامه.

مثلا كل البرامج التي تستخدم هذه المكتبة يجب أن تقوم بإنشاء كائن الشاشة "Screen Object" وهو الكائن المسؤول عن ما يحدث على الشاشة وعملية إنشائه كما يلي :

1- في ملف التعاريف لبرنامجك (في أمثلة الكتاب نستخدم الملف "main.h")
نقوم بكتابة التعريف التالي :

```
AGDXScreen* Screen;
```

2- بعد ذلك في برنامجنا (في أمثلة الكتاب نستخدم الملف "main.cpp") نقوم بإنشاء الكائن كما يلي :

```
Screen = new AGDXScreen();
```


نحن الآن قمنا بإنشاء الكائن Screen ونستطيع أن نستغل جميع أعضائه ولكن مع هذا الكائن بالذات لازالت هناك عدة خطوات يجب أن نقوم بها أولاً.

3- وبعد ذلك نقوم باختيار نوع شاشة البرنامج "باستخدام نافذة من نوافذ Windows أو الشاشة بكاملها" كما يلي :

Screen->CreateFullScreen(hwnd, 640, 480, 8);

في هذه الحالة اخترنا الشاشة بكاملها مع "8" بت للألوان (هذا الرقم يعني 2^8 أو 256 لون). لا تقلق كل هذه الخطوات ستكون أوضح في الأمثلة. نحن هنا فقط نحاول توضيح سهولة إنشاء الكائنات واستخدامها في برامجنا.

4- بعد ذلك نقوم باستخدام ملف الألوان المطلوبة للمشهد (نقوم بهذه العملية فقط عند استخدام 256 لون أو أقل) كما يلي :

Screen->LoadPalette("أسم الصورة المحتوية على ملف الألوان");

طبعا نقوم بكل هذه الخطوات مرة واحدة فقط عند بداية البرنامج.

5- وأخيرا عند نهاية برنامجنا يجب أن نقوم بتحرير الذاكرة لهذا الكائن كما يلي :

delete Screen;

وجميع الكائنات الأخرى تتبع نفس الطريق تقريبا.

الفصل السادس



أمثلة المكتبة AGDX

طبعا المثال هو من أهم عوامل التعرف على كل موضوع جديد في عالم الحاسب الآلي. وسوف نقوم بعدة أمثلة لنوضح سهولة استخدام مكتبة AGDX. لكن قبل الدخول في أمثلة AGDX قد تسأل عزيزي القارئ مدى فعالية هذه المكتبة لاستخدامها في إنتاج برامج ذات مستوى تجاري، وللتأكد من فاعلية هذه المكتبة انظر إلى اللعبة العربية "فضاء". فقد صُممت باستخدام هذه المكتبة. ولمن يتساءل عن كيفية استعمال هذه المكتبة لتكنولوجيا DirectX أو لمعرفة كيفية استخدام هذه التكنولوجيا بشكل عام فقد خصص الجزء الثاني من هذا الكتاب لتفاصيل عمل DirectX وكيفية التعامل معها عن طريق هذه المكتبة المستخدمة. ولكن عزيزي القارئ إذا كنت من المبتدئين فإنك لا تحتاج إلى معرفة تفاصيل عمل هذه المكتبة، واستخدامها هو كل ما يهمنا في هذا الجزء من الكتاب. ولأكون صريحا، أنت قد لا تحتاج إلى قراءة الجزء الثاني من الكتاب لتقوم ببرمجة برامج الصوت والصورة وبرامج الألعاب، فكل ما تحتاجه هو استخدام مكتبة AGDX فقط. وطبعا هناك من يبحث دائما عن المزيد ليتعلم (كذلك قراءة الجزء الثاني سوف تعطينا القدرة على تطوير هذه المكتبة ولتفي بشكل أدق بما نحتاجه).

عن طريق الكثير من الأمثلة سوف نقوم باستغلال إمكانيات مكتبة AGDX كما أننا سوف نستخدم مثال مختلف لكل إمكانية وذلك بقصد التوضيح. وعن طريق استغلال إمكانيات هذه المكتبة يمكننا إنتاج برامج رائعة على جهاز الحاسب الآلي.

ملاحظة : يفترض الشرح التراكمي في الأمثلة، بمعنى أننا نفترض عند قراءتك لمثال معين أنك قد أطلعت وفهمت الأمثلة التي تسبقه.

المثال الأول

المثال الأول - صورة على الشاشة - كاملة.

في المثال الأول سنقوم عرض صورة على شاشة الكمبيوتر هذا كل ما سوف نقوم به. ولنقوم بهذه العملية سوف نقوم بـ :

1- إنشاء كائن الشاشة **Screen** وحجز الذاكرة له عن طريق الأمر **New**

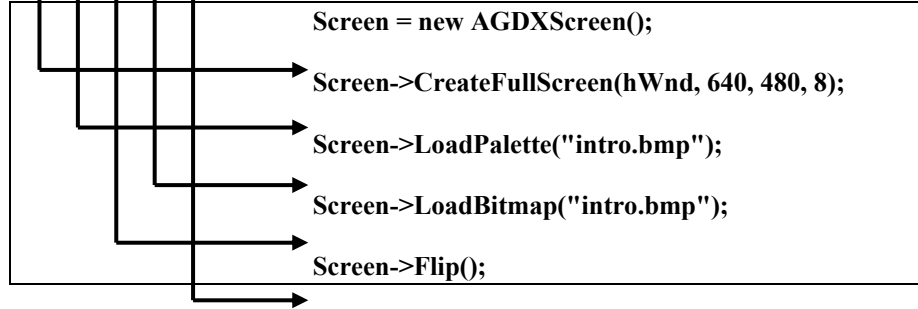
2- اختيار عدد الألوان المستخدم ونقاء الشاشة وإذا كانت عبارة عن نافذة أو شاشة كاملة

3- تعبئة لوحة الألوان التي سنقوم باستخدامها.

4- تعبئة الصورة المطلوب عرضها في السطح الخلفي.

4- قلب محتويات السطح الخلفي إلى السطح الأمامي لنتمكن من رؤية الصورة.

الآن لنري كيف نقوم بذلك في برنامجنا :



نعم هذا كل ما في الأمر ، ما تبقى من البرنامج هو عبارة عن برنامج **win32** والذي سبق واطلعنا عليه.

في المثال الأول نقوم بشرح البرنامج خطوة بعد خطوة وذلك لقصد التوضيح ، أما فيما بعد في الأمثلة التي تليه سنكتفي بشرح ما هو جديد في كل منها.

ملاحظة : جميع الأمثلة موجودة على القرص المدمج (CD-ROM) ويمكنك تجربتها مباشرة أو تعبئتها على برنامج Visual C++.

خطوات إنشاء المثال الأول :

1- أولاً نقوم بتعبئة برنامج win32 (البرنامج موجود على القرص المدمج في الملف win32 ولمعرفة تفاصيل كيفية كتابة برنامج win32 راجع الفصل الثالث).

2- نقوم بإضافة تعريف المكتبة وكائن الشاشة كما يلي إلى أعلى البرنامج:

```
#include <AGDX.h>
AGDXScreen* Screen;
```

3- ليس هناك أي تغيير في الخطوة الأولى ، الثانية ، الثالثة ، الرابعة والخامسة من برنامج win32

4- في الخطوة السادسة نقوم بإضافة ما سبق وشرحناه عن كائن الشاشة:

```
Screen = new AGDXScreen();
Screen->CreateFullScreen(hWnd, 640, 480, 8);
Screen->LoadPalette("intro.bmp");
Screen->LoadBitmap("intro.bmp");
Screen->Flip();
```

5- نزيل جميع محتويات الخطوة التاسعة وذلك لأننا لم نعد في حاجة لاستخدام Device context (DC) هو ما يستخدمه النظام Windows قبل إدخال تكنولوجيا

DirectX (في الحقيقة لا يزال يُستخدم في البرامج التي لا تستخدم تكنولوجيا DirectX) لأننا نستخدم سطوح (Surfaces) العنصر DirectDraw أحد عناصر تكنولوجيا DirectX.

6- نزيل الرسالة WM_PAINT ومحتوياتها لأننا أزلنا الـ Device context (DC) واستخدمنا DirectDraw بدل منها وبما أننا عند استخدام DirectDraw يجب أن نقوم بأجراء التغييرات المطلوبة بأنفسنا لم تعد هناك حاجة لهذه الرسالة من رسائل Windows (في حالة Device context (DC) تستخدم هذه الرسالة للتجديد الشاشة)

هذا كل ما في الأمر، لقد قمنا سويةً بكتابة برنامج كامل يستخدم تكنولوجيا DirectX ويقوم بإظهار صورة على الشاشة ، الصورة موجودة من ضمن المثال ويمكن طبعا استخدام صورة مختلفة لتجربتها. أو بالأحرى عليك استخدام صور مختلفة لترى النتيجة بنفسك. لا تنسى أن تغيير اسم الصورة (intro.bmp) إذا كنت تريد استخدام صور مختلفة (يجب كذلك أن تكون من نوع bmp).

برنامج المثال الأول

```

////////////////////////////////////
// example 1 - Fullscreen applications
// You must link to AGDX.lib, ddraw.lib and dxguid.lib
////////////////////////////////////

#include <windows.h>
#include <windowsx.h>
#include <AGDX.h>

AGDXScreen* Screen;// The AGDXScreen object, every program must have one

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

//Start of WinMain
int WINAPI WinMain (
    HINSTANCE hInstance,           // handle to current instance
    HINSTANCE hPrevInstance,     // handle to previous instance
    PSTR szCmdLine,              // pointer to command line
    int iCmdShow                 // show state of window
)
{
    // STEP #1 - Set up your variables  الخطوة "1" إعداد المتغيرات

```

```

WNDCLASSEX wndclass;
MSG         msg;
HWND        hWnd;           //Windows program handle
static char szAppName[] = "AGDX example 1";

```

```
// STEP #2 - Set Elements of a "WNDCLASSEX" structure
الخطوة "2" إعداد أعضاء المتغير "WNDCLASSEX" من نوع struct
```

```
//
```

```
wndclass.cbSize      = sizeof (wndclass);
wndclass.style       = CS_HREDRAW | CS_VREDRAW;
wndclass.lpfnWndProc = WndProc; //<---- Very Important. See Step #8
wndclass.cbClsExtra  = 0;
wndclass.cbWndExtra  = 0;
wndclass.hInstance   = hInstance;
wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION);
wndclass.hIconSm     = LoadIcon (NULL, IDI_APPLICATION);
wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW);
wndclass.hbrBackground= (HBRUSH) GetStockObject (WHITE_BRUSH);
wndclass.lpszMenuName= NULL;
wndclass.lpszClassName = szAppName;
```

```
// STEP #3 - Register the "WNDCLASSEX" تسجيل "3" الخطوة
```

```
if (!(RegisterClassEx (&wndclass)))
    return FALSE;
```

```
// STEP #4 - Now you can actually create the window
```

```
// الخطوة "4" إنشاء نافذة البرنامج
```

```
hWnd = CreateWindowEx (
    WS_EX_APPWINDOW,      //Extended Window Style
    szAppName,           //Window Class Name
    szAppName,           //Window Caption
    WS_OVERLAPPEDWINDOW, //Window Style
    CW_USEDEFAULT,       //Initial X Pos
    CW_USEDEFAULT,       //Initial Y Pos
    CW_USEDEFAULT,       //Initial X Size
    CW_USEDEFAULT,       //Initial Y Size
    NULL,                //Parent Window Handle
    NULL,                //Window Menu Handle
    hInstance,           //Program Instance Handle
    NULL);               //Creation Paramaters
```

الخطوة "5" إظهار نافذة البرنامج على الشاشة // STEP #5 - Display the window

```
ShowWindow (hWnd, iCmdShow);
```

الخطوة "6" تجديد النافذة // STEP #6 - Force an update to the window

```
UpdateWindow (hWnd);
```

```
// Create the AGDXScreen object and set the resolution
Screen = new AGDXScreen();
```

```
// Set the Screen resolution and nubmer of colors
Screen->CreateFullScreen(hWnd, 640, 480, 8);
```

```
// Load the palette from the tiles bitmap
Screen->LoadPalette("intro.bmp");
```

```
// Load the Bitmap to the back buffer
Screen->LoadBitmap("intro.bmp");
```

```
// Flip the back buffer to the front
Screen->Flip();
```

// Step #7 - The WinMain function enters an infinite loop which is referred to // as a windows "message pump"

الخطوة "7" تدخل وظيفة WinMain() دورة الرسائل والتي تستمر لنهاية البرنامج//

```
while (TRUE)
{
```

```
    if (!GetMessage (&msg, NULL, 0, 0))
        return msg.wParam;
```

```
    TranslateMessage (&msg);
    DispatchMessage (&msg);
```

```
}
```

```
}
//End of WinMain
```

```
// Step #8 - The windows callback function "WinProc"
```

```
//           الخطوة "8" وظيفة WinProc
```

```
//Start of WndProc
```

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT iMsg, WPARAM  
wParam, LPARAM lParam)
```

```
{
```

```
// Step #1 WinProc - Define variables           الخطوة "9" تعريف المتغيرات
```

```
// Step #2 WinProc - The switch statement for handling windows messages
```

```
//           الخطوة "10" الأمر الشرطي switch لدورة الرسائل
```

```
switch (iMsg)
```

```
{
```

```
case WM_KEYDOWN :
```

```
switch (wParam)
```

```
{
```

```
case VK_ESCAPE: //Exit on ESC or F12 keys
```

```
case VK_F12:
```

```
PostMessage (hWnd, WM_CLOSE, 0, 0);
```

```
return 0;
```

```
}
```

```
return 0;
```

```
//Basically its the proper way to exit a program
```

```
case WM_DESTROY :
```

```
SAFE_DELETE(Sound);
```

```
PostQuitMessage (0);
```

```
return 0;
```

```
}
```

```
return DefWindowProc (hWnd, iMsg, wParam, lParam);
```

```
}
```

```
//End of WndProc
```


المثال الثاني

المثال الثاني – صورة على الشاشة – نافذة.

في هذا المثال سنعمل على تحويل المثال الأول والذي يقوم بعرض صورة على الشاشة كاملة وذلك بالقيام بنفس العملية ولكن هذه المرة مع استخدام نافذة من نوافذ Windows ولكي نرى سهولة هذه العملية كل ما علينا هو أن نستخدم الأمر التالي في الخطوة العاشرة من البرنامج السابق :

```
Screen->CreateWindowed(hWnd, 640, 480);
```

بدل الأمرين السابقين :

```
Screen->CreateFullScreen(hWnd, 640, 480, 8);
Screen->LoadPalette("intro.bmp");
```

لأن الأمر الثاني "LoadPalette("intro.bmp")" يقوم بتعبئة لوحة الألوان وبما أننا نتعامل الآن مع نافذة من نوافذ Windows فإننا لا نحتاج لذلك لأننا سوف نستخدم ملف ألوان Windows.

هذا كل ما نحتاج عملة لتغيير البرنامج ليتحول إلى نافذة من نوافذ Windows ، ولكن نوافذ Windows لها مميزات لا نستطيع أن نتجاهلها وبالأخص الميزتين التاليتين :

- نستطيع تحريك نوافذ Windows على الشاشة
- نستطيع تغيير حجمها

لذلك علينا أن نقوم ببعض التغييرات الإضافية لبرنامجنا لوضع هذين الميزتين في الاعتبار. أولاً ماذا يحدث عندما نقوم بتحريك أو تغيير حجم نافذة برنامجنا؟ لو قمنا فقط بتحويل المثال الأول لاستخدام النافذة كما شرحنا أعلاه، فإننا عند تحريك أو تغيير حجم النافذة فسوف تختفي الصورة من الشاشة. وذلك لأننا قمنا بأمر البرنامج بإظهار الصورة على النافذة ولكننا لم نخبره ماذا يفعل لو أن هناك تغييراً حصل على النافذة لذلك علينا القيام بالخطوتين الإضافيتين التاليتين :

1- نُضيف أمر التوقيت التالي إلى الخطوة الخامسة :

```
SetTimer(hWnd, 1, 100, NULL);
```

2- نضيف رسالة التوقيت إلى دورة الرسائل للبرنامج في الخطوة العاشرة :

```
case WM_TIMER :
Screen->Flip();
return 0;
```

وعندما تتلقى Windows رسالة التوقيت يقوم كائن الشاشة بقلب محتويات الشاشة الخلفية (عن طريق الأمر "Screen->Flip()" ونحن نعلم بأن الشاشة الخلفية تحتوي على الصورة لأننا سبق ووضعتها في الخطوة السادسة، راجع المثال الأول لترى بنفسك كيف تم ذلك) إلي الشاشة الرئيسية



فنرى الصورة على الشاشة وبذلك تكون الصورة دائما ظاهرة.

أمر ورسالة التوقيت :

الآن المثال الثاني اصبح متكامل (والكمال لله وحده) بمجرد تغيير عدة اسطر في المثال الأول استطعنا أن نحول برنامج كامل لاستخدام نافذة من نوافذ Windows. ولكن ما هي رسالة التوقيت ؟ ولماذا استخدمناها هنا ؟

أمر التوقيت هو الأمر "SetTimer(hWnd, 1, 100, NULL)" وهو أحد أوامر Win32 ونستخدمه لقياس الوقت في برامجنا. وعند إضافته نقوم بإضافة رسالة التوقيت كذلك "WM_TIMER" والتي يعمل البرنامج على طلبها حسب عدد المرات التي نحددها في أمر التوقيت، في هذا المثال استخدمنا القيمة "100" كل 100 ملي من الثانية يقوم البرنامج بإرسال رسالة التوقيت. (كل ثانية تحتوي على 1000 ملي من الثانية).

المثال الثالث

المثال الثالث – لتخلص من win32 وبرمجتها

اعتقد بأننا تحدثنا بما فيه الكفاية عن برمجة win32 وحين الوقت للدخول فعلا في برمجة الصوت والصورة للكمبيوتر. في هذا المثال سنقوم بتعديل المثال الأول بحيث نضع برنامج win32 في ملف خاص به ثم نقوم ببعض التعديلات عليه وذلك حتى نتوقف بشكل نهائي (طبعاً سنقوم بالتحدث كلما احتاج الأمر عن الخوض في برمجة win32 ونبدأ في التركيز على موضوع هذا الكتاب. هذا المثال سيقوم بنفس عمل المثال الأول ، ولكن هذه المرة سوف نفصل برنامج win32 في ملف خاص به ولأننا لن نحتاج لإجراء أي تعديلات جوهرية بعد هذا المثال عليه.

سنقوم أولاً بإنشاء ثلاث وظائف كما يلي :

1- الوظيفة الأولى "Start_Program ()" وهي المسؤولة عن إنشاء كائن الشاشة وكل ما هو مطلوب القيام به عند بداية البرنامج (ناقشنا إنشاء كائن الشاشة في المثال الأول ، وكل ما سنقوم به هنا هو نقل تلك الخطوات إلى هذه الوظيفة).

2- الوظيفة الثانية "Main ()" وهي مثل وظيفة "Main ()" تحت النظام DOS وفيها سنقوم بكتابة البرنامج الرئيسي.

3- الوظيفة الثالثة "Finish_Program ()" وهي المسؤولة عن كل ما هو مطلوب القيام به عند نهاية البرنامج.

طبعاً سنقوم بإنشاء ملفين إضافيين أحدهما يحمل تعاريف البرنامج المطلوب استخدمها (سنطلق عليه الاسم "main.h") والثاني نضع فيه هذه الوظائف (سنطلق عليه الاسم "Main.cpp") وسوف نلقي النظر إلى هذه العملية لاحقاً.

بعض التعديلات على الملف win32.CPP :

الآن سوف نحتاج إلى القيام بالتعديلات المناسبة في البرنامج الرئيسي والذي يحتوي على برنامج win32 كما يلي :

- أولاً نبدأ ببرنامج المثال الأول ونقوم بنقل ما يلي إلى ملف التعاريف :

```
#include <windows.h>
#include <windowsx.h>
#include <AGDX.h>
AGDXScreen* Screen؛
```

ونضع بدلها أمر النظر إلى ملف التعاريف :

```
#define Ex1_CPP
#include "main.h"
```

- في الخطوة "1" نقوم بنقل التعريف "HWND hWnd;" وهو المتغير الذي يحمل عنوان برنامجنا إلى ملف التعاريف.
- في الخطوة "4" سنقوم بتغيير أحد مميزات نافذة البرنامج من "WS_OVERLAPPEDWINDOW" إلى "WS_POPUP" لأننا نستخدم الشاشة بكاملها (نضعه كما هو في حالة تغيير البرنامج لاستخدام نافذة من نوافذ Windows) هذه الخطوة ليست ضرورية ولكنها مستحبة. لو أنك

جربت المثال الأول سوف ترى شاشة بيضاء "لجزء من الثانية" ثم تختفي ، ولذلك نحن نقوم بهذه الخطوة حتى لا نرى تلك الشاشة المزعجة مطلقاً في البرامج التي تستغل الشاشة بكاملها. (تحذير: يجب إرجاع هذه الميزة إلى حالتها الأولى في حالة أن البرنامج يستخدم نافذة ، وإلا فإنك لن ترى شيئاً على الشاشة)

- في الخطوة "6" نقوم بنقل جميع خطوات إنشاء كائن الشاشة إلى الوظيفة "Start_Program ()" ونضع بدلاً عنها أمر الذهاب إليها.
- في الخطوة "7" سنقوم ببعض التعديلات حتى نستطيع استخدام وظيفتنا الجديدة "Main ()" كما يلي:

```
while(1)
{
    if(PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))
    {
        if(!GetMessage(&msg, NULL, 0, 0)) return msg.wParam;
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    else if(bActive)
    {
        Main();
    }
    else WaitMessage();
}
```

باختصار يقوم الكمبيوتر بالذهاب إلى الوظيفة Main() كلما كان المتغير "bActive" يساوي صحيح "TRUE".

- وأخيراً في الخطو "10" نقوم بنقل خطوات رسالة إنهاء البرنامج "WM_DESTROY" إلى الوظيفة "Finish_Program ()" ونضع بدلاً عنها أمر الذهاب إليها.

في الأمثلة القادمة لن نتكلم عن خطوات برمجة win32 ، إلا إذا احتجنا إليها
 طبعا ، وكل ما سنعمله هو إلحاق هذا الملف إلى برامجنا لا أكثر.

ملف التعاريف Main.H :

في هذا الملف نقوم بوضع تعاريف البرنامج ، هذا يجعل من برامجنا أكثر سهولة
 للفهم وأقل تعقيدا ونرى الملف كما يلي :

```
#ifndef MAIN_H
#define MAIN_H

#include <windows.h>
#include <windowsx.h>
#include <AGDX.h>

#ifdef Ex1_CPP
#define DECLARE
#else
#define DECLARE extern
#endif

DECLARE AGDXScreen* Screen;
DECLARE HWND hWnd;
DECLARE BOOL bActive;

void Start_Program(void);
void Finish_Program(void);
void Main(void);

#endif
```

وفية نقلنا تعاريف البرنامج (هنا أيضا سنقوم بإضافة أي تعاريف جديدة في
 المستقبل). الأوامر "#define" و "#ifndef" و "#ifdef" و "#else" و "#endif"

وضعت هنا لقصد التأكد أن الكمبيوتر لن يقوم بتعريف المتغيرات أكثر من مرة وبالتالي عدم حدوث أية مشاكل.

الملف الرئيسي Main.cpp :

سوف تعلم عزيزي القارئ الآن لماذا قمنا بكل ذلك ، هذا لأن برنامجنا الرئيسي سيصبح سهلاً للغاية. وإذا كنت تعرف البرمجة على النظام Dos فسوف تكتشف مدى سهولة هذا البرنامج وتقاربه مع النظام القديم. كذلك لاحظ أن ملف برنامج win32 لن يتغير بشكل رئيسي من الآن فصاعداً ، ولذلك لن نهتم به كثيراً ولكن التغيير سوف يكون هنا في برنامجنا الرئيسي.

طبعا الملف الرئيسي سوف يحتوي على الوظائف الثلاث والتي سبق وأشرنا إليها ونرى الملف الرئيسي في الصفحة التالية.

نعم يا صاح ، هذه هي كل محتويات الملف الرئيسي (انظر لصفحة التالية) أو بالأحرى هذا هو كل برنامج المثال الثالث. ألا تلاحظ السهولة بعد أن تخلصنا من برمجة win32 (نحن طبعا لم نتخلص منها ولكننا وضعناها في ملف خاص بها ولن نهتم إلا بالحاق ذلك الملف في برامجنا القادمة).

وفي كل برامجنا القادمة سنعمل على تغيير الملف الرئيسي وملف التعاريف. أما ملف برنامج win32 (والموجود في الملف win32.CPP) فلن نهتم به إلا بشكل بسيط وذلك من وقت إلى آخر (لا ترتبك سأحاول أن لا أهتم به أبداً إلا عند الضرورة).

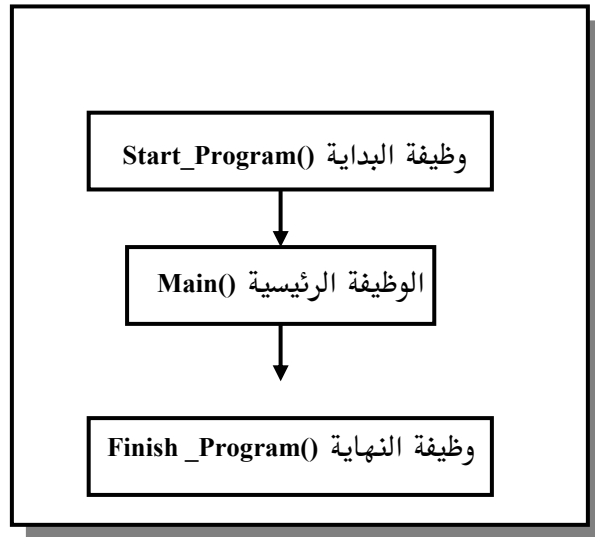
```
#include "main.h"

void Start_Program(void)
{
    bActive=TRUE;
    Screen = new AGDXScreen();
    Screen->CreateFullScreen(hWnd, 640, 480, 8);
    Screen->LoadPalette("intro.bmp");
    Screen->LoadBitmap("intro.bmp");
    Screen->Flip();
}

void Finish_Program(void)
{
    SAFE_DELETE(Sound);
    PostQuitMessage (0);
}

void Main(void)
{
}
}
```

وقد خصصنا فصل خاص للتحدث عن برمجة win32 بشكل مفصل في هذا الكتاب. وكذلك خصصنا الجزء الثاني من هذا الكتاب للأخوة والأخوات المهتمين بتفاصيل عمل تكنولوجيا DirectX. ولكن لا تقلق إذا كان كل اهتمامك هو الجزء الأول من هذا الكتاب والذي يركز على إنتاج برامج ذات مستوى ممتاز بدون الاهتمام بتفاصيل برمجة DirectX.



الشكل العام لبرامجنا

المثال الرابع

المثال الرابع - صوت ، صورة ، موسيقى و... أكشن



عزيز القارئ ، إذا كنت متابِعاً معنا رحلتنا في هذا الكتاب مع برمجة الكمبيوتر فإننا قد وصلنا إلى نقطة أصبحت الأمور فيها أكثر سهولة مما كانت عليه. وستكتشف من الآن فصاعداً أن برمجة الكمبيوتر عملٌ ممتع ، بعد أن قررنا في المثال السابق أننا قد اكتفينا بما نريد معرفته عن برمجة win32 و الدخول في موضوع الكتاب الحقيقي وهو برمجة الصوت والصورة على الكمبيوتر، أولاً بالأبعاد الثنائية (الصور المتحركة) ثم فيما بعد بالأبعاد الثلاثية. وسترى أنه بقليل من البرمجة أصبح بإمكاننا إنتاج برامج ذات مستوى ممتاز. كما أننا سوف نرى إمكانية استخدام الملفات الموسيقية من النوع "Midi" (تستطيع أن تجد موسيقى أغلب الأغاني العربية على هذا النوع من الملفات في الإنترنت) والتي تتميز بمستواها الصوتي الجيد وصغر حجمها وذلك لأن الموسيقى الموجودة عليها ليست

مسجلة وإنما عبارة عن أوامر تقوم بطاقة الصوت بعزفها ولذلك تميزت بصغر حجمها ، ولكن لنفس السبب فأن جودتها ستتخلف من بطاقة الصوت المستخدمة لأخرى. وسوف نرى كيف يمكننا استخدام الملفات الصوتية من النوع "Wave" والتي يستخدمها النظام Windows وهي عبارة عن ملفات صوتية تحتوي على تسجيل صوتي معين.

ملف التعاريف Main.H :

```

#ifndef MAIN_H
#define MAIN_H

#include <windows.h>
#include <windowsx.h>
#include <AGDX.h>

#ifdef Ex1_CPP
#define DECLARE
#else
#define DECLARE extern
#endif

DECLARE AGDXScreen* Screen;
DECLARE HWND hWnd;
DECLARE BOOL bActive;

// Sound objects
DECLARE AGDXMusic* Music; ← A
DECLARE AGDXSound* Sound; ← B
DECLARE AGDXSoundBuffer* SND_Sing; ← C

void Start_Program(void);
void Finish_Program(void);
void Main(void);
#endif

```

لا يختلف هذا الملف كثيرا عن المثال السابق فكل ما قمنا به هنا هو إضافة تعريف للكائنات :

A- كائن الموسيقى AGDXMusic

B-كائن الصوت AGDXSound

C-كائن مخزن الصوت AGDXSoundBuffer

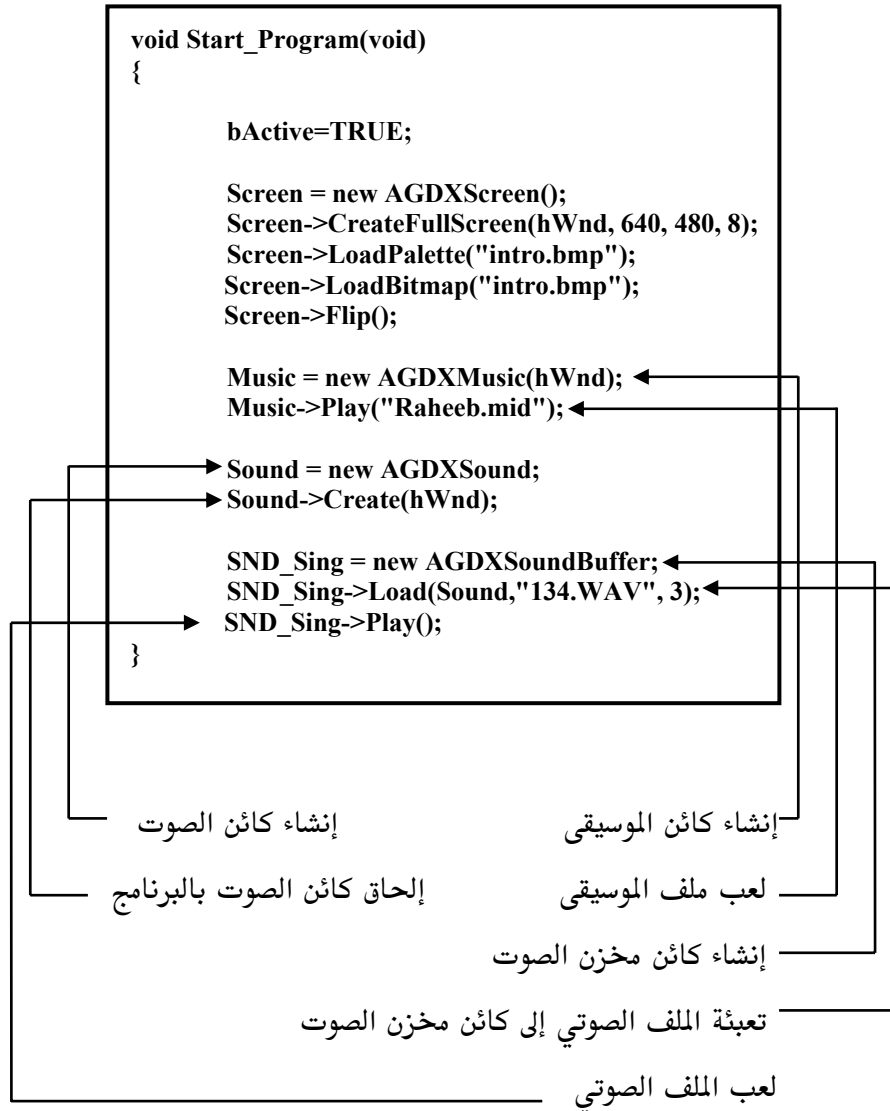
تلاحظ أننا عندما نريد استخدام أحد كائنات المكتبة AGDX نقوم بتعريف الكائن في هذا الملف (كما فعلنا في السابق لإنشاء كائن الشاشة AGDXScreen) ثم نقوم بإنشائه فيما بعد في الملف الرئيسي.

كذلك لاحظ أن جميع كائنات المكتبة AGDX تبدأ باسمها ، مثلا كائن الموسيقى يكتب هكذا "AGDXMusic" حيث نرى انه يبدأ بالكلمة "AGDX" وكذلك الحال مع بقية الكائنات لهذه المكتبة ولذلك في شرح الأمثلة أحيانا سنقوم باستخدام اسم الكائن مباشرة، مثلا كائن الموسيقى سيكون "Music" بدل "AGDXMusic" أو كائن الشاشة سنستخدم أحيانا الاسم "Screen" بدل "AGDXScreen" وهكذا مع بقية الكائنات.

بما أننا في هذا المثال نريد استعراض كيفية إنشاء واستغلال كائن الصوت وكائن الموسيقى. لهذا فقد قمنا بتعريفها هنا حتى نتمكن من استخدامها في البرنامج الرئيسي.

الملف الرئيسي Main.cpp ، وظيفة البداية (Start_Program):

وفي الملف الرئيسي كذلك ليس هناك كثير من الاختلافات عن المثال السابق وإنما بعض الإضافات لإنشاء الكائنات التي عرفناها في ملف التعاريف. لنلقي نظرة أولاً على وظيفة بداية البرنامج:



لاحظ أننا عندما أنشأنا كائن الموسيقى ، قمنا مباشرة بتشغيل الملف الموسيقي الملحق. بيد أننا في حالة الكائن الصوتي قمنا أولاً بإنشائه ثم قمنا بإلحاقه بالبرنامج ، بعد ذلك احتجنا لإنشاء الكائن المخزن للصوت والذي عن طريقه نقوم بتخزين ملف الصوت في الذاكرة. ونحن نحتاج إلى كلا الكائنين (كائن الصوت وكائن مخزن الصوت) لكي نستطيع استعمال الملفات الصوتية من نوع "wave".

وأخيراً لاحظ الرقم "3" في أمر تعبئة الملف الصوتي إلى كائن مخزن الصوت ، ويعني أننا قد نستخدم نفس الصوت في البرنامج ثلاث مرات وفي نفس الوقت (نعم ، أعرف بأنك تسأل نفسك الآن ، ماذا يقصد باستخدام نفس الصوت ثلاث مرات؟). افترض مثلاً أننا نستخدم صوت انفجار في برنامج معين ، ونتوقع حدوث ثلاثة انفجارات في نفس الوقت على الشاشة ، عندها سوف نحتاج إلى تشغيل الملف الصوتي للانفجار ثلاث مرات في نفس الوقت ، ولذلك نستعمل هذا الرقم. ولكن أنتبه ، كلما كان هذا الرقم أكبر كلما استعملنا حيز أكبر من الذاكرة الصوتية.

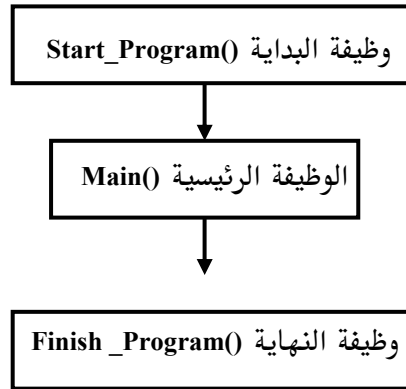
الملف الرئيسي Main.cpp ، الوظيفة الرئيسية Main(void):

في هذا المثال لن نقوم باستخدام الوظيفة الرئيسية (Main) لعدم حاجتنا لها. بالنسبة للأخوة والأخوات الذين سبق لهم برمجة النظام Dos أحب أن أوضح أن الوظيفة الرئيسية هنا تختلف قليلاً عما تعودتم عليه ، هنا يقوم الكمبيوتر باستدعاء هذه الوظيفة عشرات المرات في الثانية. بمعنى أن الكمبيوتر عندما يصل على نهاية هذا الوظيفة يذهب مرة أخرى إلى بدايتها وتكرر هذه العملية عشرات المرات في الثانية ، أما في النظام Dos فإن الكمبيوتر يقوم بالمرور مرة واحدة فقط عليها. وهي شبيهة بالوظيفة "Update()" في مكتبة الألعاب الملحقة بكتابي السابق "برمجة

الألعاب على النظام Windows95. "طبعاً هذا شيء جيد لأن كل برامج الصوت والصورة هي عبارة عن حلقة دائرية ، حيث يبدأ الكمبيوتر من نقطة معينة في البرنامج إلى أن يصل إلى نهايته ، عندها يقوم بتكرار العملية ، وفي كل مرة يواجه إدخال معين من المستعمل (مثل تحريك إشارة الفأرة أو عصا الألعاب أو الضغط على أحد أزرار لوحة المفاتيح مثلاً) فإنه يقوم بعمل معين رداً على ذلك.

الملف الرئيسي Main.cpp ، ووظيفة النهاية Finish_Program() :

وأخيراً في وظيفة النهاية تقوم بإيقاف الأصوات عن طريق الأمر Stop() و تحرير الذاكرة للكائنات التي استخدمناها عن طريق الأمر "SAFE_DELETE".



المثال الخامس

المثال الخامس- الخلفيات التطبيقية.التدكم في الإدخال

عرفنا أن أحد مميزات مكتبة AGDX هي الخلفيات المتحركة ، وهناك نوعين من الخلفيات (نعبر عنهما بكائنين) ، الخلفيات التطبيقية وخلفيات التايلز (Tiles). في هذا المثال سوف نتحدث عن كائن الخلفيات التطبيقية "AGDXLayer" وهو باختصار عبارة عن صورة بحجم الشاشة نستطيع تحريكها في كل الاتجاهات (ناقشنا النوعية الأخرى في فصل خاص بها). وتقوم مكتبة AGDX باستخدام أسطح العنصر DirectDraw لحفظ كل خلفية على سطح ، طبعاً هذه العملية تتم داخل AGDX بدون تدخل منا ولكن يجب أن نكون بعلم بها.

ملف التعاريف Main.H :

كذلك في هذا المثال سوف نستخدم كائن الإدخال "AGDXInput" وذلك للتحكم في لوحة المفاتيح (بالإضافة إلى التحكم في الفأرة و عصا الألعاب) لذلك نقوم في البداية بتعريف الكائنين : كائن الخلفية التطبيقية وكائن الإدخال في ملف التعاريف كما يلي :

```
DECLARE AGDXInput Input;
DECLARE AGDXLayer* Background;
```

(ملاحظة: بقية الملف لم تتغير عن المثال السابق)

وبعد ذلك ولكائن الإدخال فقط سوف نحتاج إلى إضافة أمر الإنشاء إلى ملف Win32 (نعم لقد سبق واتفقنا أننا سوف نقوم بتغييرات أو إضافات بسيطة إلى هذا الملف حسب الحاجة) نضيف هذا الأمر إلى الخطوة السادسة:

```
Input.Create(hInstance,hWnd);
```

والسبب في أننا أضفناه إلى ملف Win32 وليس في وظيفة بداية البرنامج "Start_Program()" (كما تعودنا عند إنشاء كائنات المكتبة AGDX) هو أننا نستخدم المتغير "hInstance" في أمر الإنشاء لهذا الكائن ، وهذا المتغير لا يوجد إلا في وظيفة "WinMain()" من وظائف برنامج Win32.

الملف الرئيسي Main.cpp ، وظيفة البداية Start_Program() :

أما في وظيفة البداية فنقوم بإنشاء كائن الخلفية التطبيقية فقط (لأننا سبق وأنشأنا كائن الإدخال في ملف Win32) كما يلي :

```
Background = new AGDXLayer(Screen, "أسم الملف.bmp");
```

(ملاحظة: بقية الوظيفة لم تتغير عن المثال السابق)

ونقوم بكتابة أسم الملف الذي يحتوي على الصورة (يجب أن تكون الصورة من نوع bmp).

الملف الرئيسي Main.cpp ، الوظيفة الرئيسية Main(void):

الآن دعونا نرى كيف يمكننا أن نستخدم لوحة المفاتيح للتحكم في حركة الخلفية إلى جهة اليمين أو إلى جهة اليسار ، سوف نستخدم الأسهم للتحكم في الحركة ونرى الملف الرئيسي كما يلي:

```
void Main(void)
{
    Input.Update();

    if(Input.Keys[DIK_RIGHT])
    {
        Background->ScrollRight(1);
    }

    if(Input.Keys[DIK_LEFT])
    {
        Background->ScrollLeft(1);
    }

    Background->Draw(Screen->GetBack());
    Screen->Flip();
}
```

- أولاً نقوم باستخدام الأمر Input.Update(); حتى يمكننا استغلال كائن الإدخال.

- بعد ذلك استخدمنا الأمر الشرطي "If" ويعني "إذا" ، إذا تم ضغط السهم المتجه إلى اليمين [DIK_LEFT] Input.Keys حرك الخلفية إلى اليمين والعكس صحيح.
- وأخيرا نستخدم الأمرين التاليين لرسم موقع الخلفية الجديد على السطح الخلفي ثم قلبه إلى السطح الرئيسي حتى نراه :

```
Background->Draw(Screen->GetBack());
Screen->Flip();
```

الملف الرئيسي Main.cpp ، ووظيفة النهاية End_Program():

أما في وظيفة نهاية البرنامج فنقوم بتحرير الذاكرة للكائنات التي استخدمناها. هذه الوظيفة أيضا مطابقة لوظيفة المثال السابق ولكن مع إضافة تحرير كائن الخلفية:

```
SAFE_DELETE(Background);
```

هذا هو كل البرنامج ، ترى عزيز القارئ سهولة البرمجة عند تعاملنا مع المكتبة AGDX وبقية الأمثلة لا تختلف شيئا عما ذكرناه إلى الآن. وربما كانت الأمور وللوهلة الأولى تبدو كثيرة ولكن سوف تكتشف بعد فترة بأن الكثير من الأمور التي سبق وشرحناها بدأت في التكرار.

المثال السادس

المثال السادس- نريد أن نرى الحركة !

في هذا المثال سوف نقدم ميزة جديدة من مميزات مكتبة AGDX وربما تكون أهم ميزة في برمجة الصوت والصورة بشكل عام وهي الحركة وبالتحديد حركة الممثلين.

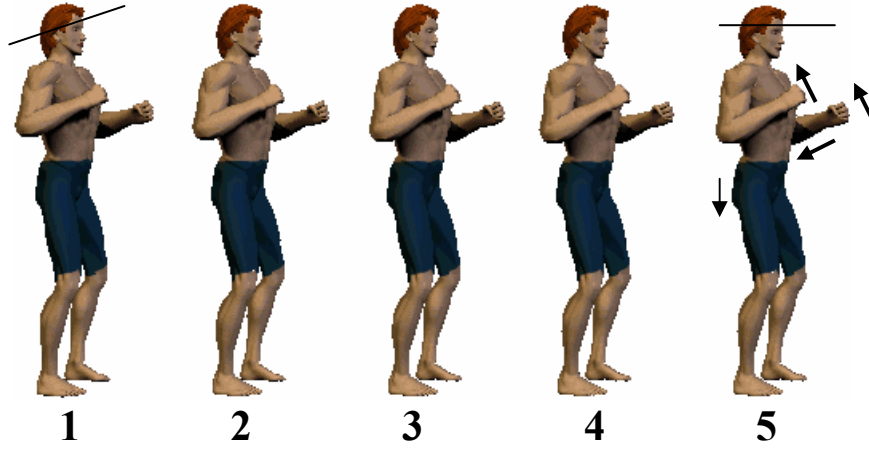
من هم الممثلين ؟ نعم تستطيع أن تقول أن برامج الصوت والصورة هي في الواقع مثل الفيلم السينمائي ، ولا يصلح أي فلم سينمائي بدون ممثلين. والممثل من وجهة نظرنا هو كل ما يتحرك على الشاشة من إنسان أو حيوان أو جماد (عندما نريد أن نستخدم "باب" لمنزل بحيث يفتح ويغلق يصبح هذا الباب ممثل وعندما نريد أن نحرك حصان من يمين الشاشة إلى يسارها يصبح هذا الحصان ممثل ونفس القاعدة تنطبق على البشر). تصور أننا نريد أن نبرمج لعبة مثل لعبة (Street Fighter 2) أو "مصارعة الشوارع" الشهيرة. عندها سوف نحتاج إلى مقاتلين ، ولجعل الأمور سهلة سوف نستخدم مقاتل أو "ممثل" واحد لهذا المثال.

القص من هذا المثال هو معرفة كيفية استخدام الممثلين في برامجنا ، وهنا نستخدم ممثل ليقوم بشخصية "الملاكم" وسيكون في إحدى حالتين: الأولى في حالة وقوف وانتظار والثانية في حالة إلقاء لكمة للخصم.



صحيح أنه في الحالة الأولى سيكون الممثل في حالة وقوف ولكن لو أننا جعلنا الممثل واقف هكذا بدون حراك لن يكون شكله مقنعا لأننا في الواقع لا يمكن أن نقف بدون أية حركة لمدة طويلة وإلا فإننا قد سلبنا أهم عنصر من عناصر إعطاء الحياة وهي الحركة. لذلك سوف نقوم بجعل الممثل يقوم بالتحرك حتى وهو في حالة الوقوف ، وسوف نعبر

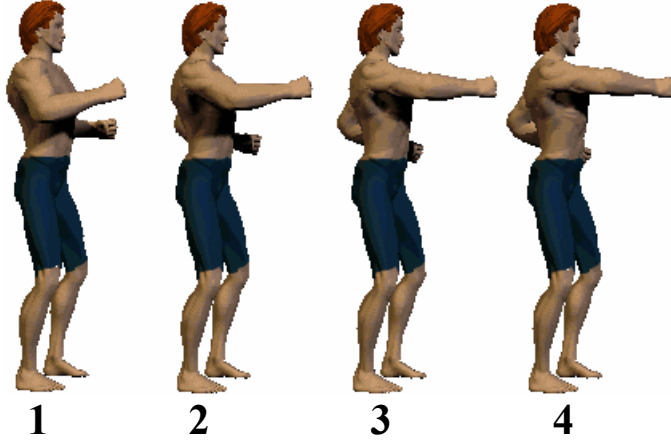
عن هذه الحركة باستخدام عدد من الصور (الإطارات) السينمائية كما يلي :



قد تتساءل عزيزي القارئ لماذا جميع هذه الإطارات "Frames" تبدو متشابهة ولكن في واقع الأمر هي ليست كذلك ، فهناك اختلاف بين كل منها (حاولت توضيح الفرق بين الإطار الأول والأخيرة باستخدام الأسهم) ولتري ذلك جرب المثال السادس من القرص المدمج.

السؤال الآخر الذي قد يأتي على البال هو كيف استطعنا إنتاج هذا الممثل؟ هل رسمناه؟ أو هل هو صورة عن ملاكم حقيقي؟ ولإجابة هذه الأسئلة لنلقي نظرة أخرى على الملاكم ، سوف ترى بأنه من إنتاج الكمبيوتر وقد استخدمت برنامج "Poser3" (بوزر ثري) لإنتاج هذه الصور (الإطارات). وهناك فصل خاص في هذا الكتاب لشرح طريقة عمل هذا البرنامج وكيفية الاستفادة منه. طبعاً أنت تستطيع إنتاج الصورة بأي أسلوب تشاء (هناك عشرات البرامج المخصصة للرسم وإنتاج الأبعاد الثلاثية وإنما اخترت هذا البرنامج بالذات لثقتي بأنه من أفضل البرامج لهذا النوع من الحركة وكذلك سعره في حدود المعقول).

أما في الحالة الثانية ، عندما يقوم الملاكم بإلقاء لكمة إلى الخصم فإننا عبرنا عنها باستخدام أربع إطارات سينمائية كما يلي :



ونستخدم ملف واحد يحمل جميع هذه الصور (أو نطلق عليها الإطارات "Frames") كما يلي :

طبعاً عند استخدام البرنامج "Poser" أو أغلب برامج إنتاج الصور على الكمبيوتر فإنها لا تعطينا جميع الإطارات المطلوبة للحركة في صورة واحدة كما نرى في هذا المثال ، وإنما نحصل على صورة مختلفة لكل إطار ولذلك نستخدم البرنامج الفريد من نوعه "Tyler 2" (يسمى تايلر 2) والذي يأتي مجاناً مع الكتاب في وضع جميع

الإطارات في صورة واحدة كما نرى في الصورة التالية:



(في هذه المثال نرى اللون الرمادي) ويسمى هذا اللون "مفتاح" أو مفتاح الألوان (Color Key) لأنه اللون الوحيد الذي لا يظهر على الشاشة عند عرض هذا الممثل.

كذلك لاحظ أن هذا اللون المحيط بكل الإطارات

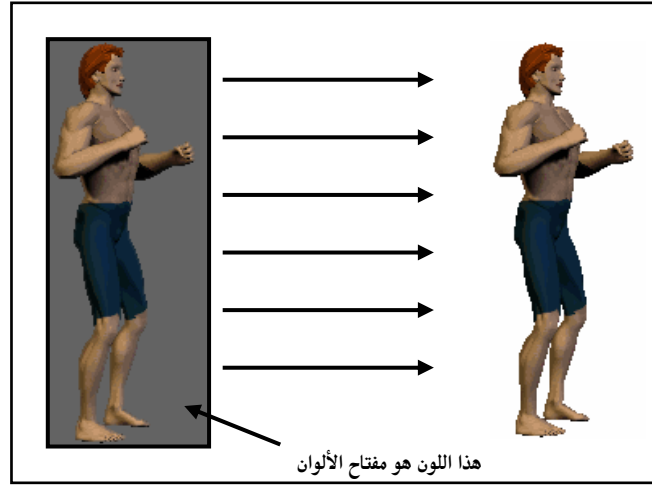
منسق الإطارات Frame Tiler

لو نظرنا إلى المثال (ROIDS) وهو أحد أمثلة المجموعة البرمجية لـ DirectX لرأينا أنه جمع كل الصور المستخدمة فيه في صورة واحدة من نوع bmp بحيث يقوم باستخدامها عند اللزوم ، وتلك الطريقة ممتازة جدا عند إنشاء برامج تستخدم الصوت والصورة على الشاشة وفي الواقع أن أغلب الألعاب وبرامج الميديا في

الأسواق تستخدم هذه الطريقة في تخزين الصور المطلوبة سواء كانت صور ثابتة أو متحركة الأشكال. ولكن كيف نجعل الصور التي سوف نستخدمها في برامجنا تستخدم نفس الطريقة؟ هناك جوابين لهذه المشكلة : الحل الأول هو أن نقوم بذلك عن طريق أحد برامج الرسم بحيث نرتب الصور الواحدة تلو الأخرى (بعد ساعات وساعات من المحاولة الناجحة أحيانا والفاشلة أحيانا أخرى سوف تكتشف مدى صعوبة هذه الطريقة) والحل الآخر هو استخدام برنامج خاص يقوم بذلك لنا ، للأسف بحثت كثيرا عن برنامج يقوم بهذا العمل ولكن دون فائدة وأخيرا وجدت شخص اسمه "DAVID ROSTOWSKY" قام بوضع برنامج يقوم بما نريد يطلق عليه Frame Tiler يقوم بهذا العمل ويمكنك تجربة هذا البرنامج (يوجد على القرص المدمج مع هذا الكتاب).

مفتاح الألوان Color Key :

ذكرنا بأن هناك لون محيط بكل إطار أطلقنا عليه "مفتاح الألوان". فعندما نقوم بإنتاج أي صورة على الكمبيوتر فإنها لابد وأن تكون إما مربعة أو مستطيلة ونحن كل ما يهمنا هو الشكل الموجود بها ولا نريد اللون المحيط بذلك الشكل أن يظهر على الشاشة. ونوضح هذه العملية في الشكل التالي ، والذي استخدمناه في هذا المثال :



وليس شرطاً أن يكون مفتاح الألوان هو اللون الرمادي فقد يكون أي لون نريده المهم هو أن نخبر كائن الممثل برقم اللون وذلك من لوحة الألوان المستخدمة للمشاهد، هذا طبعاً عند استخدامنا 8 bit (أو لوحة ألوان تتكون من 256 لون) والشاشة بأكملها. أما عند استخدامنا 16 bit (أو 16 مليون لون) أو استخدام نافذة من نوافذ Windows فإن مفتاح الألوان لابد أن يكون اللون الأسود ونعبر عنه بالقيمة "0" صفر.

ملف التعاريف Main.H :

أعتقد بأن المقدمة للحركة في هذا المثال كانت واضحة بما فيه الكفاية لذلك تعالوا معنا لنرى كيف يمكننا أن نطبق ما شرحناه في هذا المثال، وكما هي العادة في بقية الأمثلة سوف نرى كيف نقوم أولاً بتعريف كائن الممثل :

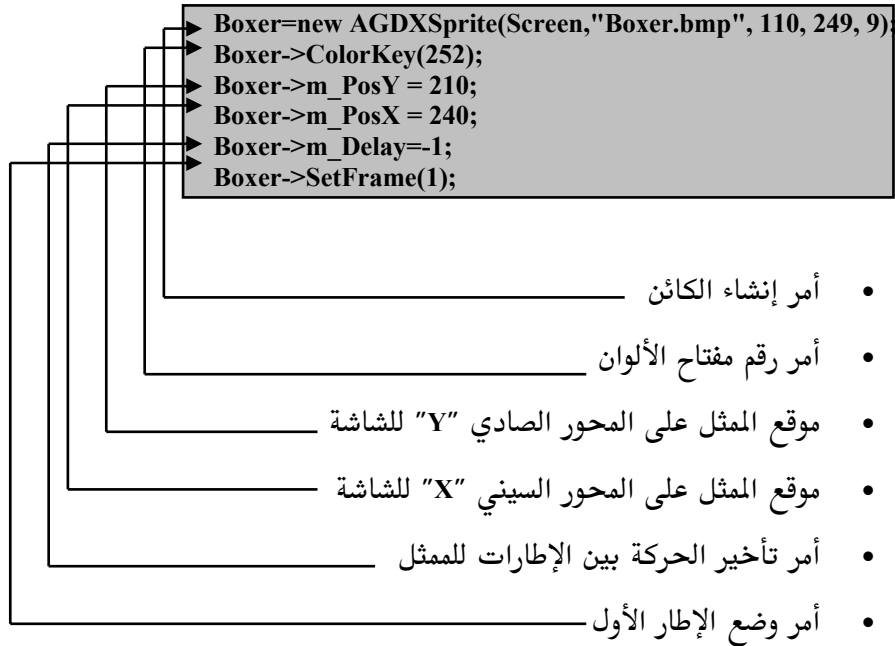
```
DECLARE AGDXSprite* Boxer;
```

كائن الممثل "AGDXSprite" يعطينا القدرة على إنشاء أي عدد من الممثلين في برامجنا. في هذا المثال أنشأنا ممثل أطلقنا عليه الأسم "Boxer". وهذا كل ما نحتاج إلى كتابته في ملف التعاريف.

أريد أن أوضح بأن هذا ليس الكائن الوحيد لإنشاء الممثلين في مكتبة AGDX وسوف نتعرف على كائن آخر في أمثلة قادمة كما أننا سنذكر الاختلافات والمميزات لكل منهما.

الملف الرئيسي Main.cpp ، وظيفة البداية (Start_Program):

أما في وظيفة البداية فنقوم بإنشاء كائن الممثل كما يلي :



في أمر إنشاء الكائن "AGDXSprite(Screen,"Boxer.bmp", 110, 249, 9)" استخدمنا الرقم "110" لنعبر عن عرض كل إطار و الرقم "249" لنعبر عن ارتفاع كل إطار للممثل ثم استخدمنا الرقم "9" لنعبر عن مجموع الإطارات التي سوف نستخدمها وأخيرا استخدمنا الاسم "Boxer.bmp" لنعبر عن اسم الملف الذي يحمل صورة (والتي بدورها تحمل كل الإطارات المطلوبة للحركة) الممثل. بعد ذلك نشاهد مجموعة من الأوامر التي يجب أن نستعملها عند إنشاء هذا الكائن (لإعطائه موقعة

على الشاشة وأول إطار سيعرض ..الخ) هذه الأوامر هي عبارة عن وظائف أعضاء للكائن ممثل ونستخدمها للتحكم في هذا الممثل.

الملف الرئيسي Main.cpp ، الوظيفة الرئيسية Main(void):

في هذه الوظيفة نبرمج أساس عمل برنامجنا ، لنلقي أولاً نظرة على البرنامج ثم نقوم بالإطلاع وشرح كل سطر:

```
void Main(void)
{
    if(--Boxer->m_Delay < 0)
    {
        static int Frame=0;
        static int Delta = -1;
        if(Frame == 0 || Frame == 4) Delta = -Delta;

        Boxer->SetFrame(Frame+=Delta);
        Boxer->m_Delay = 11;

        Background->ScrollRight(1);

        Input.Update();
        if(Input.Keys[DIK_SPACE])
        {
            static int hit=4;
            static int Delta1 = -1;
            if(hit == 4 || hit == 8) Delta1 = -Delta1;

            Boxer->SetFrame(hit+=Delta1);
            Boxer->m_Delay = 4;
        }
    }
    Background->Draw(Screen->GetBack());
    Boxer->DrawTrans(Screen->GetBack());
    Screen->Flip();
}
```

أولا هذه الأسهم ليست جزء من البرنامج وإنما وضعت هنا لتوضيح بداية كل وظيفة أو أمر (عن طريق الإشارة "{") ونهايتها (عن طريق الإشارة "}") بحيث يكون شكل البرنامج أكثر وضوحاً.

في البداية استخدمنا الأمر الشرطي "if(--Boxer->m_Delay < 0) لنتحكم في سرعة الملاكم فيقوم هذا الأمر بخصم رقم "1" من المتغير "Boxer->m_Delay" كل مرة يقوم الكمبيوتر باختبار هذا الأمر الشرطي.

بعد ذلك نرى ما يلي:

```

static int Frame=0; ← A
static int Delta = -1; ← B
if(Frame == 0 || Frame == 4) Delta = -Delta; ← C

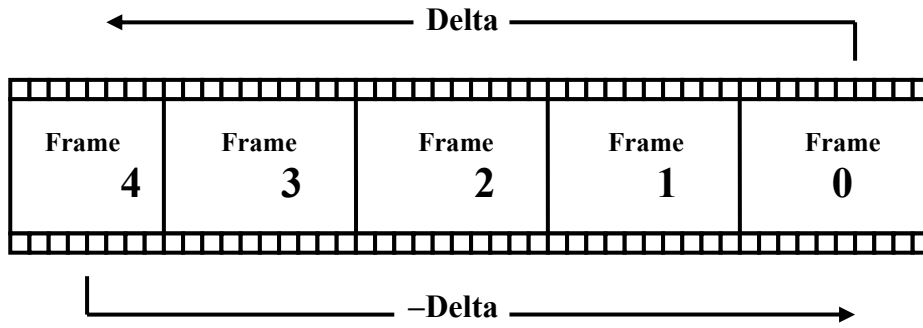
Boxer->SetFrame(Frame+=Delta); ← D
Boxer->m_Delay = 11; ← E
    
```

في "A" و "B" قمنا بإنشاء متغيرين من نوع "int". واستخدمنا الأمر "static" ويعنى أن هذين المتغيرين يحملان القيمة الابتدائية (القيمة "0" للمتغير Frame والقيمة "1-1" للمتغير Delta) مرة واحدة فقط خلال عمل البرنامج. طبعاً هذا لا يعني أن المتغيرين لن يحملان القيمة "0" أو القيمة "1-1" أبداً وإنما نستخدم الأمر "static" لنجبر الكمبيوتر على عدم استعمال أمر الإنشاء في كل مرة يمر بها على هذا الأمر،

وبالتالي عدم إعادة إنشاء المتغيرات وإعطاءها القيمة الابتدائية في كل مرة ننظر إلى أمر الإنشاء.

(أريد أن أعيد تذكيرك بأن الكمبيوتر يقوم بتنفيذ الوظيفة الرئيسية -من بدايتها إلى نهايتها ثم يعيد- مالا يقل عن ثلاثين مرة في الثانية طوال مدة تشغيل البرنامج).

في "C" استخدمنا المتغيرين السابقين في مراقبة حركة الإطارات للممثل ، فعندما يعرض الإطار الخامس يبدأ العد العكسي :



دورة الإطارات (Frames Loop)

كما هو واضح من الشكل أعلاه أن المتغير Delta يبدأ بالقيمة موجب واحد وعلية فإن المتغير Frame والذي يعبر عن الإطار الحالي للممثل يبدأ من القيمة "0" أو الإطار الأول ويزيد قيمة واحدة إلى أن يصل إلى القيمة "4" أو الإطار الخامس عندها تتغير قيمة المتغير Delta من موجب واحد إلى سالب واحد وبالتالي يبدأ العد التنازلي للمتغير Frame من القيمة "4" إلى القيمة "صفر" عندها تتغير قيمة المتغير Delta مرة أخرى إلى موجب واحد وبالتالي يبدأ العد التصاعدي للمتغير Frame و هلم جرا.

في "D" استخدمنا الأمر "Boxer->SetFrame(Frame+=Delta);" الذي يقوم باستخدام المتغيرين السابقين لعرض الإطار المطلوب، فعندما يكون المتغير "Delta" يساوي واحد فإن المتغير "Frame" يزيد كل مرة بمعدل "1" وعندما يكون المتغير "Delta" يساوي سالب واحد فإن المتغير "Frame" ينقص كل مرة بمعدل "1-".

في "E" قمنا بإعطاء المتغير "Boxer->m_Delay = 11" القيمة "11" حتى نستطيع تأخير الحركة بين كل إطار من إطارات الممثل.

بعد ذلك استخدمنا الأمر "Background->ScrollRight(1);" كما كان الحال في المثال السابق لتحريك الخلفية الطبقيّة إلى اليمين.

بعد ذلك نرى ما يلي:

```

Input.Update(); ← A
if(Input.Keys[DIK_SPACE]) ← B
{
    static int hit=4; ← C
    static int Delta1 = -1; ← D
    if(hit == 4 || hit == 8) Delta1 = -Delta1; ← E

    Boxer->SetFrame(hit+=Delta1); ← F
    Boxer->m_Delay = 4; ← G
}
    
```

في "A" وعن طريق الأمر "Input.Update();" قمنا باستدعاء كائن الإدخال لفحص أي إدخال من المستعمل سواء كان عن طريق الفأرة أو لوحة المفاتيح أو عصا الألعاب.

في "B" استخدمنا الأمر الشرطي "if" لفحص فيما إذا ضغط المستعمل على مفتاح المسافة عن طريق الأمر "Input.Keys[DIK_SPACE]".

في حالة لو أن الأمر الشرطي "if" كان صحيح فإننا في "C" و "D" و "E" و "F" و "G" نقوم بنفس العملية السابقة للتحكم في حركة الإطارات للممثل، الفرق هنا أننا نستخدم الإطارات الأربعة الأخيرة للحركة حتى نرى حركة إلقاء الكلمة.

وأخيراً نرى الأوامر التالية :

```
Background->Draw(Screen->GetBack()); ← A
Boxer->DrawTrans(Screen->GetBack()); ← B
Screen->Flip(); ← C
```

فإننا باختصار في "A" نقوم برسم الخلفية الطبقيّة على السطح الخلفي و بعد ذلك في "B" نرسم الملائم فوق الخلفية الطبقيّة أيضاً على السطح الخلفي وأخيراً في "C" نقلب محتويات السطح الخلفي إلى السطح الرئيسي لنرى النتيجة على الشاشة.

وللملاحظة أيضاً، فقد استخدمنا في هذا المثال كائن الإدخال AGDXInput وأنشأناه كباقي الكائنات. ولكي نستخدمه احتجنا إضافة الأمر التالي :

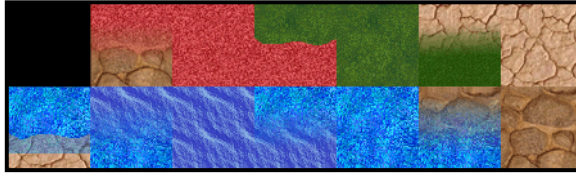
```
Input.Create(hInstance,hWnd);
```

وذلك في الخطوة السادسة من برنامج win32 بعد وظيفة البداية Start_Program() مباشرة. وعن طريقة نستطيع مراقبة الإدخال من لوحة المفاتيح أو من إشارة الفأرة أو عصا الألعاب. (سنستخدمه مرة أخرى في الأمثلة القادمة).

المثال السابع

المثال السابع- أين هي خلفيات التايلز ؟

في هذا المثال سوف نقدم أحد مميزات مكتبة AGDX وهي خلفيات التايلز، وهنا نتعرف على كيفية التعامل مع هذه النوعية من الخلفيات. تستخدم مكتبة AGDX نوعين من خلفيات التايلز ، وكلاهما يقوم بنفس العمل ، الفرق هو كيفية إنشاء الخلفيات عن طريق البرنامج الإضافي "برنامج إنشاء الخلفيات" (Tile Editor) شخصياً أنا لا أفضل البرنامج الأول (وهو البرنامج Editor) والذي نستخدمه في هذا المثال ولكنه موجود وسنتعرف عليه. لهذا السبب سوف نستخدم البرنامج "MapMaker" في المستقبل لتصميم الخلفيات. في هذا المثال سنستخدم البرنامج الأصلي لهذه المكتبة (تمت إضافة MapMaker فيما بعد).



TILES.BMP

ملف التعاريف Main.H :

سوف نرى كيف نقوم أولاً بتعريف كائن خلفيات التايلز :

```
DECLARE AGDXTile* Tiles;
DECLARE AGDXMap* Map;
DECLARE RECT Window;
```

لاحظ أننا نحتاج إلى تعريف وإنشاء كائنين لاستخدام خلفيات التايلز ، كائن خاص بصور التايلز "AGDXTile" وكائن آخر مسؤول عن خريطة توزيع هذه الصور "AGDXMap". ثم بعد ذلك قمنا بتعريف المتغير "Window" من نوع RECT والذي يعبر عن أربع نقاط لمستطيل الشاشة (نعم نستطيع تقسيم الشاشة ليكون لها أكثر من خلفية ، في لعبة مغامرات مثلاً جزء للمشاهد الرئيسي وجزء آخر لما يحمله اللاعب).

الملف الرئيسي Main.cpp ، وظيفة البداية (Start_Program):

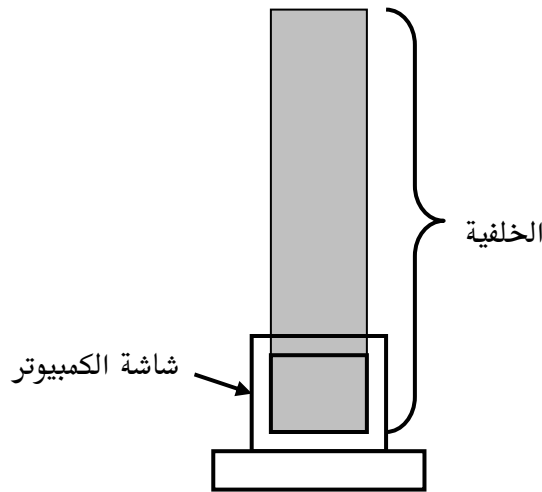
أما في وظيفة البداية فنقوم بما يلي :

```
Tiles = new AGDXTile(Screen, "TILES.BMP", 64, 64, 13);
Map = new AGDXMap(Tiles, Screen);
Map->Load("map.MAP");
SetRect(&Window, 0, 0, 640, 480);
```

أولا بإنشاء كائن التايلز "Tiles" والذي مررنا له الصورة المحتوية على التايلز " TILES.BMP" وكذلك حجم (64 للطول و 64 للعرض) كل صورة تايل وعددها (13 صورة في هذه الحالة).

بعد ذلك قمنا بإنشاء كائن الخريطة "Map" والذي مررنا له كائن التايلز (الذي يحتوي على الصور) وكائن الشاشة (الذي سنقوم بالرسم إليه).
بعد ذلك قمنا بتعبئة ملف الخريطة "map.MAP" في كائن الخريطة.

وأخيرا قمنا بإعطاء المتغير "Window" نقاطه الأربع والتي تعبر عن حجم الشاشة (نحتاج إلى هذا المتغير لنخبر كائن الخريطة بالمكان الذي يجب فيه أن يعرض الخلفية المتحركة ، لأننا نستطيع أن نقسم الشاشة إلى أكثر من قسم كل منها يحمل خلفية متحركة خاصة به. في هذا المثال استخدمنا خلفية واحدة فقط لتعرض على الشاشة بأكملها).



الملف الرئيسي Main.cpp ، الوظيفة الرئيسية Main(void):

الآن دعونا نرى كيف يكون شكل الوظيفة الرئيسية:

```
void Main(void)
{
    Map->WrapScrollUp(1);
    Map->DrawClipped(Screen->GetBack(), &Window);
    Screen->Flip();
}
```

هذا ليس جزء من الوظيفة الرئيسية ، بل هذه هي الوظيفة بأكملها.

بكل بساطة:

- نقوم بأمر كائن الخريطة بأن يحرك الخلفية نقطة واحدة كل مرة إلى الأعلى.
- بعد ذلك أمرنا كائن الخريطة بأن يرسمها في السطح الخلفي وفي المستطيل "Window" والذي سبق وعرفناه.
- وأخيرا قلب محتويات السطح الخلفي إلى السطح الرئيسي لنرى النتيجة.

الملف الرئيسي Main.cpp ، وظيفة النهاية End_Program():

أما في وظيفة نهاية البرنامج كما هي العادة مع بقية الأمثلة فإننا نقوم بتحرير الذاكرة للكائنات التي استخدمناها والخروج من البرنامج:

```
void Finish_Program(void)
{
    delete Music;
    delete Map;
    delete Tiles;
    delete Screen;

    PostQuitMessage (0);
}
```

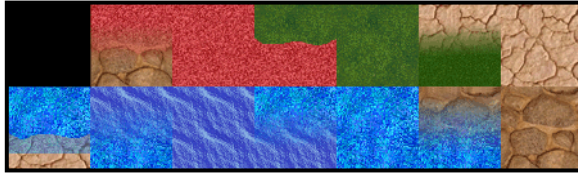
لاحظ كذلك أن ترتيب تحرير الكائنات من الذاكرة "يجب" أن يكون بشكل عكسي ، بمعنى أن الكائن الذي إنشأ أولاً في وظيفة البداية يحرر آخرًا في هذه الوظيفة.

المثال الثامن

المثال الثامن- النوع الثاني من خلفيات التايلز ؟

هذا المثال لا يختلف كثيراً عن المثال السابق وسوف نقوم كذلك بتقديم نفس ميزة خلفيات التايلز أحد مميزات مكتبة AGDX ولكن ببعض الاختلافات البسيطة.

كما عرفنا من المثال السابق أن مكتبة AGDX تستخدم نوعين من خلفيات التايلز ، وكلاهما يقوم بنفس العمل ، الفرق هو كيفية إنشاء الخلفيات عن طريق البرنامج الإضافي "برنامج إنشاء الخلفيات" (Tile Editor) وكما أوضحنا سابقاً أنني أفضل البرنامج "MapMaker" لتصميم الخلفيات وفي هذا المثال سوف نستخدم هذا البرنامج لإعطاء نفس نتيجة المثال السابق ، ولو أنك جربت هذا المثال لن ترى أي اختلاف في النتيجة النهائية.



TILES.BMP

ملف التعاريف Main.H :

سوف نرى كيف نقوم أولاً بتعريف كائن خلفيات التايلز :

```
DECLARE AGDXTile* Tiles;
DECLARE AGDXMap* Map;
DECLARE RECT Window;
```

لاحظ أننا لم نقم بأي تغيير عن المثال السابق. واحتجنا إلى تعريف وإنشاء كائنين لاستخدام خلفيات التايلز ، كائن خاص بصور التايلز "AGDXTile" وكائن آخر مسؤول عن خريطة توزيع هذه الصور "AGDXMap". ثم بعد ذلك قمنا بتعريف المتغير "Window" من نوع RECT والذي يعبر عن أربع نقاط لمستطيل الشاشة (نعم نستطيع تقسيم الشاشة ليكون لها أكثر من خلفية ، في لعبة مغامرات مثلاً جزء للمشهد الرئيسي وجزء آخر لما يحمله اللاعب).

الملف الرئيسي Main.cpp ، وظيفة البداية (Start_Program):

أما في وظيفة البداية فنقوم بما يلي :

```

Tiles = new AGDXTile();
    if (!(Tiles->CreateFromTLE(Screen, "tiles.tle" )))
    {
        MessageBox(NULL, "problem", "Dbg", MB_OK);
    }

Map = new AGDXMap(Tiles, Screen);
Map->Create(13, 70, 0x10);
Map->MoveTo(0,0);

MMMap = new MapMakerMap;
if (!MMMap->Create("land.map"))
{
    MessageBox(NULL,"War nix", "dbg", MB_OK);
}
MMMap->FillAGDXMap(Map, 0);
delete(MMMap);

SetRect(&Window, 0, 0, 640, 480);

```

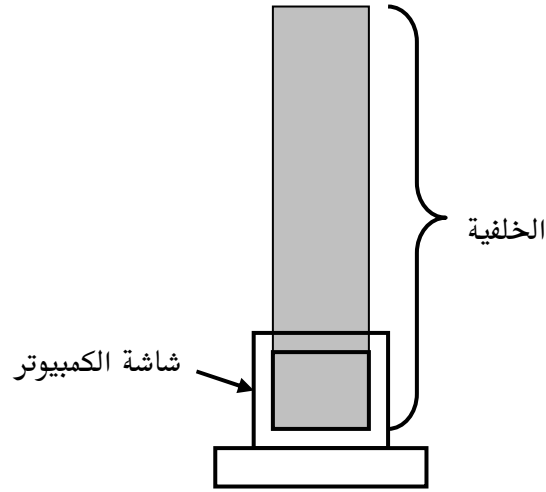
أولا بإنشاء كائن التايلز " Tiles " والذي مررنا له الصورة المحتوية على التايلز " tiles.tle " لاحظ أن هذه النوعية هي النوعية الخاصة ببرنامج تصميم صور التايلز MapMaker أما في المثال السابق فإننا استخدمنا نوعية الملفات " TILES.BMP ".

ثانيا قمنا بإنشاء كائن الخريطة " Map " والذي مررنا له كائن التايلز (الذي يحتوي على الصور) وكائن الشاشة (الذي سنقوم بالرسم إليه).

بعد ذلك أنشأنا عن طريق الأمر Create حجم الخلفية ومررنا عدد التايلز المستخدمة (70 للطول و 13 للعرض).

بعد ذلك قمنا بإنشاء كائن مؤقت لقراءة خريطة برنامج تصميم صور التايلز "MapMakerMap" والذي مررنا إليه الملف "land.map" ويقوم هو بدوره بتعبئة هذه الخريطة في كائن الخريطة والذي أنشأناه في الخطوة السابقة.

وأخيرا قمنا بإعطاء المتغير "Window" نقاطه الأربع والتي تعبر عن حجم الشاشة (نحتاج إلى هذا المتغير لنخبر كائن الخريطة بالمكان الذي يجب فيه أن يعرض الخلفية المتحركة ، لأننا نستطيع أن نقسم الشاشة إلى أكثر من قسم كل منها يحمل خلفية متحركة خاصة به. في هذا المثال استخدمنا خلفية واحدة فقط لتعرض على الشاشة بأكملها).



الملف الرئيسي Main.cpp ، الوظيفة الرئيسية Main(void):

الآن دعونا نرى ما يكون عليه شكل الوظيفة الرئيسية:

```
void Main(void)
{
    Map->WrapScrollUp(1);
    Map->DrawClipped(Screen->GetBack(), &Window);
    Screen->Flip();
}
```

هذا ليس جزء من الوظيفة الرئيسية ، بل هذه هي الوظيفة بأكملها.
بكل بساطة:

- نقوم بأمر كائن الخريطة بأن يحرك الخلفية نقطة واحدة كل مرة إلى الأعلى.
- بعد ذلك أمرنا كائن الخريطة بأن يرسمها على السطح الخلفي (عن طريق كائن الشاشة Screen->GetBack() حصلنا على السطح الخلفي) وفي المستطيل "Window" الذي سبق وعرفناه.
- وأخيرا قلب محتويات السطح الخلفي إلى السطح الرئيسي لكي نستطيع أن نرى النتيجة.

لاحظ هنا عدم اختلاف هذه الوظيفة هنا عنها في المثال السابق.

الملف الرئيسي Main.cpp ، وظيفة النهاية End_Program():

أما في وظيفة نهاية البرنامج كما هي العادة مع بقية الأمثلة فإننا نقوم بتحرير الذاكرة للكائنات التي استخدمناها والخروج من البرنامج:

```
void Finish_Program(void)
{
    delete Music;
    delete Map;
    delete Tiles;
    delete Screen;

    PostQuitMessage (0);
}
```

لاحظ كذلك أن ترتيب تحرير الكائنات من الذاكرة "يجب" أن يكون بشكل عكسي ، بمعنى الكائن الذي أنشئ أولاً في وظيفة البداية يحرر آخرًا في هذه الوظيفة.

المثال التاسع

المثال التاسع- الخلفيات المستطيلة

في هذا المثال سوف نقوم كذلك باستخدام خلفيات التايلز ، ولكن بشكل مختلف. سوف نستخدم صورة لخلفية واحدة Panoramic (باناراميك وهي الصور المستطيلة التي غالبا ما تنتهي ببدايتها). ليس من الممكن استخدام كائن الطبقات لأن هذه النوعية من الصور تكون طويلة عرضيا وتكنولوجيا دايركت اكس لا تسمح باستخدام صور ذات مساحة اكبر من عرض الشاشة ، والحل هنا هو تقسيم هذه الخلفية إلى صور التايلز (تستطيع أن تستخدم برنامج Tyler للقيام بهذه العملية) ثم نستخدم برنامج تركيب صور التايلز (MapMaker) لإعادة تركيب هذه الصور الصغير لتعطينا الخلفية مرة أخرى. وأخيراً نقوم باستخدام صور التايلز هذه كما فعلنا في الأمثلة السابقة. لنرى ذلك بالتفصيل :



الخلفية المستطيلة (Panoramic)

كما نرى أن الخلفية هي عبارة عن صورة عرضها 2400 نقطة بيد أن الشاشة في هذا المثال عرضها 600 x 800 نقطة.

ملف التعاريف Main.H :

سوف نرى كيف نقوم أولاً بتعريف كائن خلفيات التايلز :

```
DECLARE AGDXTile* Tiles;  
DECLARE AGDXMap* Map;  
DECLARE RECT Window;
```

لاحظ أننا لم نقوم بأي تغيير عن المثالين السابقين واحتجنا إلى تعريف وإنشاء كائنين لاستخدام خلفيات التايلز ، كائن خاص بصور التايلز "AGDXTile" وكائن آخر مسؤول عن خريطة توزيع هذه الصور "AGDXMap". بعد ذلك قمنا بتعريف المتغير "Window" من نوع RECT والذي يعبر عن أربع نقاط لمستطيل الشاشة (مرة أخرى تذكر أننا نستطيع تقسيم الشاشة ليكون لها أكثر من خلفية ، في لعبة مغامرات مثلاً قسم للمشهد الرئيسي وقسم آخر لما يحمله اللاعب).

الملف الرئيسي Main.cpp ، وظيفة البداية Start_Program():

أما في وظيفة البداية فنقوم بما يلي :

```
bActive=TRUE;
Screen = new AGDXScreen();
Screen->CreateFullScreen(hWnd, 800, 600, 8);
Screen->LoadPalette("lobby.bmp");

Tiles = new AGDXTile();
if (!(Tiles->CreateFromTLE(Screen, "lobby.tle" )))
{
    MessageBox(NULL, "... uh-oh", "Dbg", MB_OK);
}

// Create and load the map
Map = new AGDXMap(Tiles, Screen);
Map->Create(20,5, 0x10);
Map->MoveTo(0,0);

MMMap = new MapMakerMap;
if (!MMMap->Create("lobby.map"))
{
    MessageBox(NULL, "War nix", "dbg", MB_OK);
}
MMMap->FillAGDXMap(Map, 0);
delete(MMMap);

SetRect(&Window, 0, 0, 800, 600);

Music = new AGDXMusic(hWnd);
Music->Play("Raheeb.mid");
```

أولا وكما هو الحال مع بقية الأمثلة نقوم بوضع المتغير bActive يساوي "True" أي صحيح ثم قمنا بتحديد صفاء الشاشة النقطي ليكون 800 x 600 نقطة وبعدها ألوان 8 بت (أي $256 = 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2$ لون).

بعد ذلك استخدمنا الأمر "Screen->LoadPalette("lobby.bmp");" لتعبئة الألوان لكائن الشاشة. لا حظ أننا لا نريد الصورة lobby.bmp وإنما نريد الألوان التي تحويها، لذلك يمكننا أن نجعلها أصغر من حجمها الطبيعي بكثير لأنها لن تفقد ملف الألوان إذا فعلنا ذلك وعندها يكون حجمها اصغر بكثير.

بعد ذلك نقوم بإنشاء كائن التايلز "Tiles" والذي مررنا له الصورة المحتوية على التايلز "lobby.tle" لاحظ أن هذه النوعية هي النوعية الخاصة ببرنامج تصميم صور التايلز MapMaker.

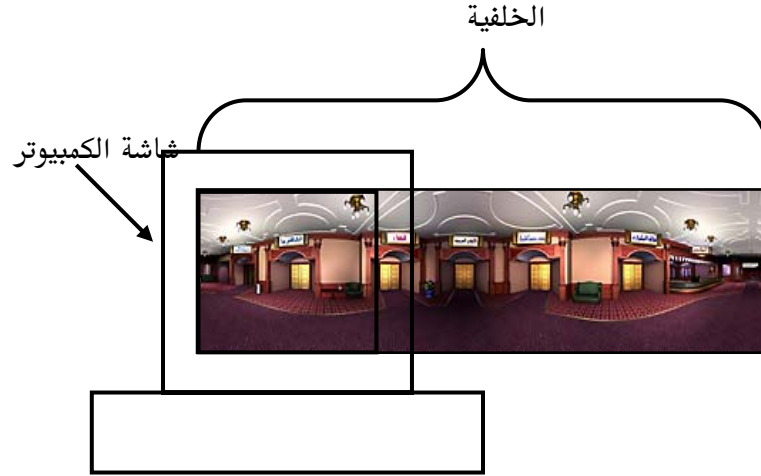
ثم قمنا بإنشاء كائن الخريطة "Map" والذي مررنا له كائن التايلز (الذي يحتوي على الصور) وكائن الشاشة (الذي سنقوم بالرسم إليه).

بعد ذلك أنشأنا عن طريق الأمر Create حجم الخلفية ومررنا عدد التايلز المستخدمة (5 للطول و 20 للعرض).

بعد ذلك قمنا بإنشاء كائن مؤقت لقراءة خريطة برنامج تصميم صور التايلز "MapMakerMap" والذي مررنا إليه الملف "lobby.map" ويقوم هو بدوره بتعبئة هذه الخريطة في كائن الخريطة والذي أنشأناه في الخطوة السابقة.

وأخيرا قمنا بإعطاء المتغير "Window" نقاطه الأربع والتي تعبر عن حجم الشاشة (نحتاج إلى هذا المتغير لنخبر كائن الخريطة بالمكان الذي يجب فيه أن يعرض الخلفية المتحركة ، لأننا نستطيع أن نقسم الشاشة إلى أكثر من قسم كل منها

يحمل خلفية متحركة خاصة به. في هذا المثال استخدمنا خلفية واحدة فقط لتعرض في الشاشة بأكملها).



الملف الرئيسي Main.cpp ، الوظيفة الرئيسية Main(void):

الآن دعونا نرى كيف يكون شكل الوظيفة الرئيسية:

```
void Main(void)
{
    Map->WrapScrollLeft(1);
    Map->DrawClipped(Screen->GetBack(), &Window);
    Screen->Flip();
}
```

هذا ليس جزء من الوظيفة الرئيسية ، بل هذه هي الوظيفة بأكملها.
بكل بساطة:

- نقوم بأمر كائن الخريطة بأن يحرك الخلفية نقطة واحدة كل مرة إلى اليسار.
- بعد ذلك أمرنا كائن الخريطة بأن يرسمها في السطح الخلفي وفي المستطيل "Window" والذي سبق وعرفناه.
- وأخيرا قلب محتويات السطح الخلفي إلى السطح الرئيسي لنرى النتيجة.

لاحظ أيضاً عدم اختلاف الوظيفة هنا عنها في المثال السابق.

الملف الرئيسي Main.cpp ، وظيفة النهاية End_Program():

أما في وظيفة نهاية البرنامج كما هي العادة مع بقية الأمثلة فإننا نقوم بتحرير الذاكرة للكائنات التي استخدمناها والخروج من البرنامج:

```
void Finish_Program(void)
{
    delete Music;
    delete Map;
    delete Tiles;
    delete Screen;

    PostQuitMessage (0);
}
```

لاحظ كذلك أن ترتيب تحرير الكائنات من الذاكرة "يجب" أن يكون بشكل عكسي ، بمعنى الكائن الذي إنشأ أولاً في وظيفة البداية يحرر آخرًا في هذه الوظيفة.

المثال العاشر

المثال العاشر - أين هي خلفيات التايلز ؟ مرة أخرى

هذا المثال مطابق للمثال السابع من حيث الشفرة البرمجية ولكن يختلف في أننا سنقوم باستخدام خلفية تم إنتاجها على البرنامج MapMaker. ولو لاحظت كذلك أننا استخدمنا هذا البرنامج مع المثال الثامن ولكننا قمنا باستخدام كائن خاص به.

في هذا المثال ليس هناك داعي لاستخدام الكائن الخاص بالبرنامج MapMaker لأننا سنقوم بتحويل الملفات الناتجة عن هذا البرنامج إلى نوعية الملفات التي يفهمها كائن الخلفية التايلز مباشرة.

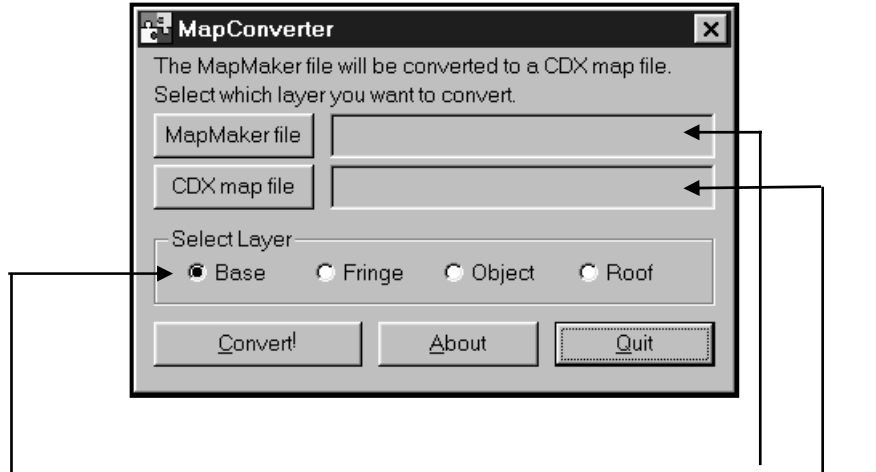
نقوم أولاً بإنشاء الخلفية المطلوبة عن طريق البرنامج MapMaker (هذه الطريقة تسمح لنا باستخدام أي عدد من الألوان لخلفيات التايلز ولكن كيف ؟) لكي نستطيع أن نستخدم صور التايلز مع البرنامج MapMaker فإننا يجب أن نحولها إلى صورة تحتوي فقط على 256 لون ثم نحولها إلى نوعية الصور التي يفهما MapMaker (راجع الفصل الرابع لمعلومات أكثر عن استخدام هذا البرنامج) ولكننا في هذا المثال نستطيع أن نستخدم صور التايلز باستخدام أي عدد من الألوان (لا تنسى أن تحتفظ بنسخة لملف صور التايلز قبل تحويله لنوعية البرامج التي يفهمها MapMaker وقبل استخدام 256 لون حتى تستطيع استخدامه فيما بعد

بملايين الألوان) وكمثال سنقوم باستخدام الخلفية من المثال الثامن وهي عبارة عن المقيين :

- الملف land.bmp والمحتوي على صور التايلز. وهذه النوعية من الملفات يمكن إنشائها عن طريق أي برنامج رسم ويمكن أن تحتوي على أي عدد من الألوان. ونحن عندما أردنا استخدامها في برنامج MapMaker فأنا حولناها عن طريق البرنامج Autograb.exe إلى الملف land.tle (
- الملف land.map المحتوي على الخريطة للخلفية والتي أنشأناها باستخدام البرنامج MapMaker وباستخدام الملف land.bmp.

ما نقوم به في هذا المثال هو أننا لن نستخدم الملف land.tle والتابع لبرنامج MapMaker ، ولكننا سنستخدم الملف الأصلي land.map بكل ألوانه. بعد ذلك سنحول الملف land.map وهو ملف الخريطة الناتج عن البرنامج MapMaker إلى الملف land.map (يحمل نفس الاسم ولكنه مختلف في التركيب) والمتوافق مع كائن صور التايلز للمكتبة AGDX. وذلك عن طريق استخدام برنامج اسمه (Map Converter) هذا الملف صمم في الأصل للمكتبة CDX ولكننا سنستطيع استخدامه في مكتبتنا AGDX بسبب التوافق للمكتبتين.

باستخدام البرنامج Map Converter سنقوم بتحويل الملف الناتج إلى نوعية الملفات للمكتبة كما يلي :



هنا نضع مكان الملف الناتج عن طريق برنامج MapMaker والذي ينتهي بالأحرف map (عن طريق الضغط على الزر MapMaker file المحاذي له).

هنا نضع مكان واسم الملف الذي نريده ، والمتوافق مع كائن التايلز للمكتبة AGDX والذي ينتهي كذلك بالأحرف map (عن طريق الضغط على الزر CDX map file المحاذي له).

بما أن كائن التايلز في المكتبة AGDX لا يتعامل إلا مع طبقة واحدة لكل ملف نختار هنا الطبقة التي نود تحويلها (طبعا نستطيع أن نحول كل الطبقات الأربع في ملفات منفصلة لكل طبقة)

كل ما نحتاج فعله الآن هو الضغط على الزر Convert! للحصول على الملف المطلوب.

ملف التعاريف Main.H :

سوف نرى كيف نقوم أولاً بتعريف كائن خلفيات التايلز :

```
DECLARE AGDXTile* Tiles;
DECLARE AGDXMap* Map;
DECLARE RECT Window;
```

لاحظ أننا نحتاج إلى تعريف وإنشاء كائنين لاستخدام خلفيات التايلز ، كائن خاص بصور التايلز "AGDXTile" وكائن آخر مسؤول عن خريطة توزيع هذه الصور "AGDXMap". ثم بعد ذلك قمنا بتعريف المتغير "Window" من نوع RECT والذي يعبر عن أربع نقاط لمستطيل الشاشة (نعم نستطيع تقسيم الشاشة ليكون لها أكثر من خلفية ، في لعبة مغامرات مثلاً جزء للمشاهد الرئيسي وجزء آخر لما يحمله اللاعب).

الملف الرئيسي Main.cpp ، وظيفة البداية (Start_Program):

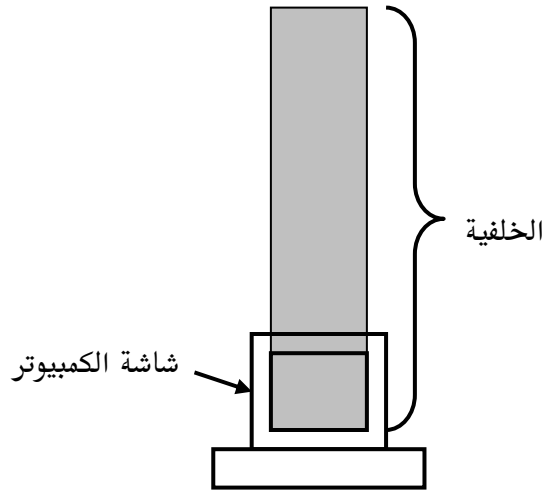
أما في وظيفة البداية فنقوم بما يلي :

```
Tiles = new AGDXTile(Screen, "land.bmp", 64, 64, 14);
Map = new AGDXMap(Tiles, Screen);
Map->Load("land.map");
SetRect(&Window, 0, 0, 640, 480);
```

أولا بإنشاء كائن التايلز "Tiles" والذي مررنا له الصورة المحتوية على التايلز "land.bmp" وكذلك حجم (64 للطول و 64 للعرض) كل صورة تايل وعددها (14 صورة في هذه الحالة).

بعد ذلك قمنا بإنشاء كائن الخريطة "Map" والذي مررنا له كائن التايلز (الذي يحتوي على الصور) وكائن الشاشة (الذي سنقوم بالرسم إليه).
بعد ذلك قمنا بتعبئة ملف الخريطة "land.map" في كائن الخريطة.

وأخيرا قمنا بإعطاء المتغير "Window" نقاطه الأربع والتي تعبر عن حجم الشاشة (نحتاج إلى هذا المتغير لنخبر كائن الخريطة بالمكان الذي يجب عليه أن يعرض فيه الخلفية المتحركة ، لأننا نستطيع أن نقسم الشاشة إلى أكثر من قسم كل منها يحمل خلفية متحركة خاصة به. في هذا المثال استخدمنا خلفية واحدة فقط لتعرض على الشاشة بأكملها).



الملف الرئيسي Main.cpp ، الوظيفة الرئيسية Main(void):

الآن دعونا نرى كيف يكون شكل الوظيفة الرئيسية:

```
void Main(void)
{
    Map->WrapScrollUp(1);
    Map->DrawClipped(Screen->GetBack(), &Window);
    Screen->Flip();
}
```

هذا ليس جزء من الوظيفة الرئيسية ، بل هذه هي الوظيفة بأكملها.
بكل بساطة:

- نقوم بأمر كائن الخريطة بأن يحرك الخلفية نقطة واحدة كل مرة إلى الأعلى.
- بعد ذلك أمرنا كائن الخريطة بأن يرسمها في السطح الخلفي وفي المستطيل "Window" والذي سبق وعرفناه.
- وأخيرا قلب محتويات السطح الخلفي إلى السطح الرئيسي لنرى النتيجة.

الملف الرئيسي Main.cpp ، وظيفة النهاية End_Program():

أما في وظيفة نهاية البرنامج كما هي العادة مع بقية الأمثلة فإننا نقوم بتحرير الذاكرة للكائنات التي استخدمناها والخروج من البرنامج:

```
void Finish_Program(void)
{
    SAFE_DELETE(Music);
    SAFE_DELETE(Map);
    SAFE_DELETE(Tiles);
    SAFE_DELETE(Screen);

    PostQuitMessage (0);
}
```

لاحظ كذلك أن ترتيب تحرير الكائنات من الذاكرة "يجب" أن يكون بشكل عكسي ، بمعنى الكائن الذي إنشأه أولا في وظيفة البداية يحرر أخرا في هذه الوظيفة.

المثال الحادي عشر

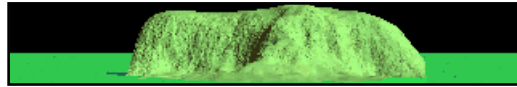
المثال الحادي عشر- الخلفيات الطبقيّة (AGDXLayer)



في هذا المثال سوف نتعرف على كيفية التعامل مع الخلفيات الطبقيّة وقبل أن أبدأ في كتابته كنت أفكر في خلفيّة تعطي الدليل على إمكانية هذه المكتبة فقامت برسم ثلاث طبقات (باستخدام البرامج Bryce 3D) :



الطبقة الأولى هي طبقة السماء (sky)



الطبقة الثانية هي طبقة الجبل الخلفي (mountain)

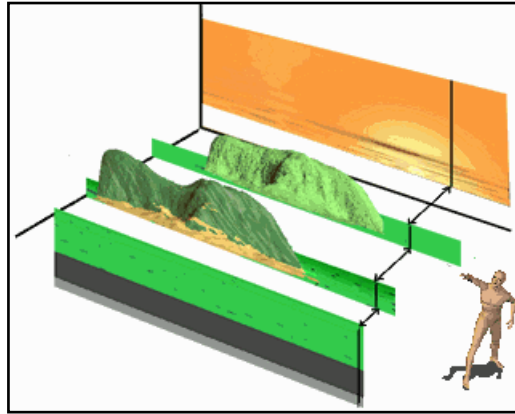


الطبقة الثالثة هي طبقة الجبل الأمامي (mountain1)



الطبقة الرابعة هي طبقة الأرضية (ground)

والآن لماذا قسمنا خلفيتنا إلى طبقات ؟ فعلنا ذلك حتى نستطيع تحريك كل طبقة على حدة وبذلك نعطي العمق الثلاثي الأبعاد المطلوب. وتلاحظ أن الطبقة الثانية والثالثة يحيط بها اللون الأسود وهو اللون المفتاح (أو يعرف كذلك باللون الشفاف) وهو اللون الغير ظاهر عند استخدام تلك الطبقة.



وتشترط المكتبة AGDX عند استخدام الطبقات في أن يكون عرض كل منها يساوي العرض المستخدم للشاشة وفي هذا الدرس نحن استخدمنا العرض 640 نقطة.

ولكن ما هي المميزات للخلفية التطبيقية ؟ عند استخدام الخلفية التطبيقية نستطيع أن نحرك تلك الطبقة بشكل ملفوف بحيث نستطيع تحريكها إلى اليمين أو اليسار فإذا اختلف جزء منها من طرف الشاشة فإنه يظهر من الجهة الأخرى وبذلك تظهر الطبقة وكأنها لا نهائية.

بعدها انتهينا من القسم المرئي للبرنامج لنبدأ بالقسم البرمجي وباستخدام المكتبة **AGDX** لننظر أولاً إلى الإضافات في ملف التعاريف **Main.h** كما يلي :

```
DECLARE AGDXScreen* Screen;
DECLARE AGDXMusic* Music;
DECLARE AGDXLayer* sky;
DECLARE AGDXLayer* ground;
DECLARE AGDXLayer* mountain1;
DECLARE AGDXLayer* mountain2;
```

ونرى أننا قمنا بتعريف الكائنات التي سنستخدمها في برنامجنا:

❖ الكائن **Screen** وهو الممثل لكائن الشاشة **AGDXScreen** (كل برامج المكتبة

تحتاج لهذا الكائن)

❖ الكائن **Music** وهو الممثل لكائن الموسيقى **AGDXMusic** (عن طريقة

نستطيع عزف الملفات من نوع **Midi**)

❖ وأربع كائنات طبقية كما رأيناها تمثل الكائن **AGDXLayer** وهي

1- كائن **sky** ممثل لطبقة السماء

2- كائن **mountain1** ممثل لطبقة الجبل الخلفي

3- كائن **mountain2** ممثل لطبقة الجبل الأمامي

4- كائن **ground** ممثل لطبقة الأرضية

أرجو أن يكون الشرح واضحاً ، و خلاصة القول أننا عندما نحتاج أن نستخدم ميزة من ميز هذه المكتبة نقوم بإنشاء متغير نعطيه اسم معين يكون ممثل للكائن المسؤول عن هذه الميزة. مثلا ميزة الخلفيات التطبيقية هي مسؤولية الكائن AGDXLayer وبالتالي قمنا بإنشاء أربع ممثلين لهذا الكائن كل منهم عبارة عن خلفية تطبيقية. وتنطبق هذه القاعدة على بقية الكائنات الأخرى في هذه المكتبة.

ثم لاحظ كذلك أن جميع الكائنات تبدأ بإسم المكتبة AGDX يتبعها عادة نوع الكائن ، في هذه الحالة مثلاً Layer ويعني طبقة بمعنى أن هذا هو كائن الطبقات أو الخلفيات التطبيقية. ولمعرفة جميع الكائنات الموجودة في هذه المكتبة يمكنك أن تلقي نظرة إلى ملف التعاريف التابع لها **AGDX.h**. (أنظر إلى الفصل الخاص لشرح إمكانية كل كائن وعمل كل وظيفة فيه في هذا الكتاب).

```
#include "main.h"

// This is our main program

void Start_Program(void)
{
  bActive=TRUE;

  // Create the AGDXScreen object and set the resoultion
  Screen = new AGDXScreen();

  // Set the Screen resolution and nubmer of colors
  Screen->CreateFullScreen(hWnd, 640, 480, 8);

  // Load the palette from the tiles bitmap
  Screen->LoadPalette("sky.bmp");
```

```
// lets first load all the layers
sky      = new AGDXLayer(Screen,"sky.bmp");
ground   = new AGDXLayer(Screen,"ground.bmp");
mountain1 = new AGDXLayer(Screen,"m1.bmp");
mountain2 = new AGDXLayer(Screen,"m.bmp");

// lets set the Key Color
mountain1->ColorKey(0);
mountain2->ColorKey(0);

//Creat the AGDXMusic object and use the window handler
Music = new AGDXMusic(hWnd);
Music->Play("furelise.mid");
}

void Finish_Program(void)
{
// we are finished so lets Delete all the objects and free the memory
SAFE_DELETE(sky);
SAFE_DELETE(ground);
SAFE_DELETE(mountain1);
SAFE_DELETE(mountain2);
SAFE_DELETE(Music);
SAFE_DELETE(Screen);

// now we can just exit the program
PostQuitMessage (0);
}

void Main(void)
{
// ----to control the time-----
firstTick = timeGetTime();
diffTick = firstTick - lastTick;
lastTick = firstTick;
WaitTime += diffTick;
// -----

if (WaitTime > (20) )
{
mountain1 ->ScrollRight(2);
mountain2 ->ScrollRight(3);
ground    ->ScrollRight(5);

WaitTime =0;
}
}
```



```

sky      ->Draw(0,0,Screen->GetBack());
mountain1 ->DrawTrans(0,190,Screen->GetBack());
mountain2 ->DrawTrans(0,200,Screen->GetBack());
ground   ->Draw(0,317,Screen->GetBack());

//Ok, now flip the Screen
Screen->Flip();
}

```

وكما نرى الآن في الأعلى أن هذا هو برنامجنا الأساسي فنلقني نظرة لوظائفه
الثلاث :

❖ أولاً وظيفة الإعداد أو البداية (**Start_Program()**)

في هذه الوظيفة سوف نقوم بإعداد الكائنات (أو بالأحرى ممثلي الكائنات) التي عرفناها في ملف التعريف السابق. وهي تعبر عن الأوامر التي نحب أن نمليها على جهاز الحاسب لأدائها. أولاً قمنا بإعداد كائن الشاشة :

```

Screen = new AGDXScreen();
Screen->CreateFullScreen(hWnd, 640, 480, 8);

```

وهو الكائن الرئيسي وكل برامج هذه المكتبة لابد أن تقوم بإعداده أولاً وقبل أي كائن آخر لأننا سنستخدمه لإنشاء الكائنات الأخرى. وعن طريقه نقوم بإعطاء البعد المطلوب للشاشة. في هذه الحالة 640x480 ومن ثم عدد الألوان التي نحب أن نستخدمها وهنا استخدمنا 8 بت أي 256 لون (الرقم 8 بت يدل على أن عدد الألوان عبارة عن الرقم 2 مرفوع للأس 8 ونستطيع كذلك استخدام 16 بت أو 24 بت أو 32 بت بمجرد تغيير الرقم عند إنشاء هذا الكائن).

```
Screen->LoadPalette("sky.bmp");
```

بعد ذلك عن طريق كائن الشاشة نقوم بتعبئة ملف الألوان (عند استخدام 256 لوان يجب أن نقوم بتعبئة الألوان التي سنستخدمها حتى نستطيع إظهارها بالشكل الصحيح على الشاشة) ونقوم بتعبئة ملف الألوان عن طريق إعطاء الصورة التي تحتوي على ذلك الملف "sky.bmp".

وكما تلاحظ أننا وبأوامر بسيطة نعطي مميزات برنامجنا للحاسب ليقيم بتنفيذها , ثم نقوم بإعداد الكائنات الطبقيّة كما يلي :

```
sky      = new AGDXLayer(Screen,"sky.bmp");
ground   = new AGDXLayer(Screen,"ground.bmp");
mountain1 = new AGDXLayer(Screen,"m1.bmp");
mountain2 = new AGDXLayer(Screen,"m.bmp");
```

وكما تلاحظ أننا عند إنشاء كل طبقة نقوم بإعطائها ملف الصورة المحتوي عليها و اسم الكائن المسؤول عن الشاشة (Screen). ونحن نعرف من الشرح السابق أن الطبقتين المحتويتين على صور الجبال بهما اللون المفتاح وهو اللون الذي لا نريد أن نراه ونريده أن يصبح شفافاً. ونحن كذلك نعرف انه اللون الأسود , واللون الأسود دائماً يحمل القيمة صفر. لذلك يجب علينا أن نعطي الأوامر لتلك الطبقات بوضع هذه الحقيقة في الحسبان وكما يلي :

```
mountain1->ColorKey(0);
mountain2->ColorKey(0);
```

أخيراً في وظيفة البداية قمنا بإنشاء وإعداد كائن الموسيقى , ثم مررنا له الملف الموسيقي المطلوب وهو من نوع Midi كما يلي :

❖ ثانياً وظيفة النهاية (End_Program)

هذه الوظيفة واضحة من اسمها فكل ما نقوم به هو تحرير الذاكرة التي قمنا بحجزها للكائنات في وظيفة البداية ونستخدم الأمر (SAFE_DELETE) أحد أوامر المكتبة AGDX لكل كائن سبق أنشأناه للقيام بهذه العملية. وأخيراً نستخدم الأمر (0) PostQuitMessage للخروج من البرنامج.

❖ ثالثاً و أخيراً الوظيفة الرئيسية (Main)

كما نعرف أن هذه الوظيفة هي قلب البرنامج. وفيها نقوم بالعمليات الرئيسية لبرنامجنا. فوظيفة الإعداد قامت فقط بإعداد الكائنات لنا ووظيفة النهاية قامت بالتأكد من أن كل شيء سيتم على ما يرام عند انتهاءنا من البرنامج. ولكن هنا في الوظيفة الرئيسية سوف نبدأ في التعامل مع أوامر الكائنات للمكتبة AGDX والتي قمنا بأعدادها مسبقاً للقيام بما نريد. ونشرحها بالتفصيل :

◀ أولاً استخدمنا الأمر (timeGetTime) (أحد أوامر win32) للحصول على الوقت في تلك اللحظة , ونحتاج لذلك حتى نعطي برنامجنا التوقيت المطلوب بحيث لا يكون سريع ولا بطئ وبدون أن يتأثر بسرعة المعالج بل يكون تأثره بالوقت الحقيقي.

◀ ثانياً استخدمنا الأمر (ScrollRight) أحد أوامر الكائن AGDXLayer للفت الطبقة من جهة اليمين وبالسرع المطلوبة (استخدام الرقم 1 مع هذا الأمر يعني لف نقطة واحدة لكل وحدة زمنية و 2 يعني نقطتين وهكذا).

◀ ثالثاً وبعد أن أصبحنا جاهزين لرسم تلك الخلفيات الطبقيّة للسطح الخلفي للذاكرة عن طريق الأوامر التالية (والترتيب مهم) :

```
sky      ->Draw(0,0,Screen->GetBack());
mountain1 ->DrawTrans(0,190,Screen->GetBack());
mountain2 ->DrawTrans(0,200,Screen->GetBack());
ground   ->Draw(0,317,Screen->GetBack());
```

وكما تلاحظ أننا استخدمنا أمر الرسم (السطح المطلوب الرسم فيه, x, y) `Draw` وهذا يقوم الأمر بعملية الرسم بدون الأخذ في الاعتبار اللون الشفاف أما الأمر الثاني وهو (السطح المطلوب الرسم فيه, x, y) `DrawTrans` فيقوم بنفس العملية ولكن بأخذ اللون الشفاف في الاعتبار ولهذا السبب يكون الأمر الأول أسرع بقليل طبعاً.

◀ رابعاً وأخيراً نقوم بقلب مكونات السطح الخلفي للذاكرة للسطح الأمامي والمرئي على الشاشة عن طريق الأمر `.Screen->Flip();`

بهذا نكون قد انهينا المثال الحادي عشر من أمثلة المكتبة AGDX , وفي الحقيقة أن هذا المثال قد يبدو طويلاً للوهلة الأولى ولولا الشرح لما احتجنا إلا لصفحة واحدة لكتابته . وقد استطعنا وبأوامر بسيطة ومحددة أن نقوم بعملية تستغرق عشرات الأسطر البرمجية وساعات من البرمجة والفضل يرجع لاستخدام المكتبة في اختصار كل ذلك.

المثال الثاني عشر

المثال الثاني عشر
الملفات السينمائية والكائن **AGDXAvi**

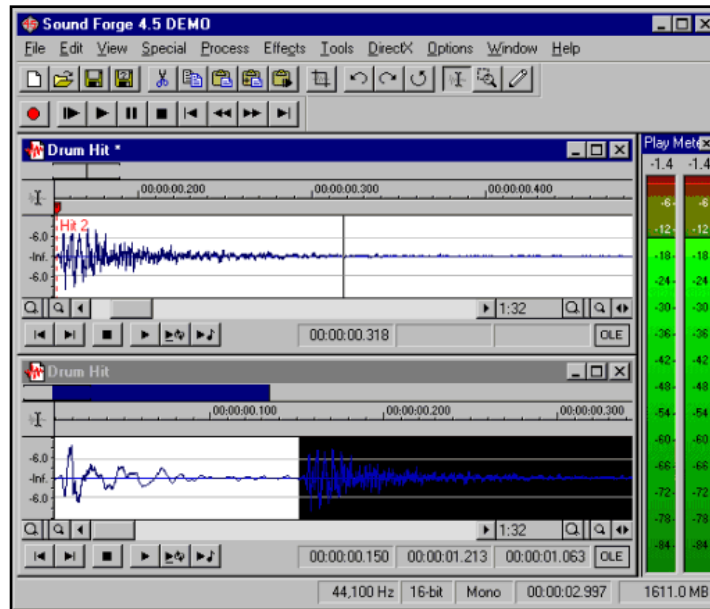


لا يكاد أي برنامج يستخدم الصوت والصورة تحت أنظمة النوافذ إلا ويستعمل الملفات السينمائية بطريقة أو أخرى. وهناك أنواع كثيرة من الملفات السينمائية مثلا الملفات ذات النوعية MPEG2 والتي تستخدم لإنتاج الأفلام على اسطوانات DVD والنوع Avi (نستطيع أن نغير أي نوع إلى آخر عن طريق بعض البرامج المجانية) وأنواع أخرى كثيرة الفرق بينها هو الحجم ومقدار نقاء الصورة. هذا الكائن وكما هو واضح من اسمه يقوم بالتعامل مع النوعية Avi وهي نوعية الأفلام التي يستخدمها النظام Windows والتي يمكن إنتاجها كذلك في اغلب برامج الأبعاد الثلاثية.

هنا في المكتبة AGDX نتعامل مع هذه الملفات السينمائية وكأنها أحد طبقات DirectX بمعنى أنه ليس بإمكاننا فقط أن نستعملها للمقدمة أو للمقاطع السينمائية

بين المشاهد لإعطاء البرنامج نوع من الحياة ولكن يمكننا كذلك استخدامها في داخل المشهد ذاته. وبذلك نستطيع دمج أي ملف فيديو داخل أي مشهد نشاء وبالطريقة التي تعجبنا كما لو أنها طبقة مثلما رأينا استخدام الطبقات في الدرس السابق.

هناك ملاحظة بالنسبة لملفات **Avi** والمكتبة **AGDX** وهي ان الملف السينمائي **يجب** ان يحتوي على قناة صوتية , او بعبارة اخرى ليس شرطاً ان يكون هناك صوت مع الملف السينمائي ولكن يشترط وجود قناة صوتية مصاحبة. ولو كانت لا تحتوي على شيء. مثلاً أنا استخدم برنامج **Sound_Froge** (تستطيع الحصول على النسخة التجريبية من www.sonicfoundry.com) ثم استخدم الأمر **Silence –insurt** لأدخال قناة صوتية غير مسموعة للملف السينمائي وهناك الكثير من البرامج التي تقوم بدمج الصوت مع هذه الملفات والتي تستطيع استخدامها. كذلك بعض البرامج ثلاثية الأبعاد مثل **Poser** أو **3D_Studio_Max** والذي عن طريقة تستطيع ادخال الصوت الى ملفات **Avi**.



Sound Forge 4.5

في هذا الدرس سوف نستخدم ملف فيديو (طبعا من نوع Avi) يحتوي على مقطع سينمائي معين (انظر اليه في بداية هذا الدرس). ثم نقوم باستخدام هذا الملف كمقدمة سينمائية للبرنامج. وفي هذا الدرس نريد ان نبرهن بساطة استخدام مميزات المكتبة AGDX بشكل عام واستخدام الملفات السينمائية بشكل خاص. ولجعل الأمور اكثر بساطة نقوم باستخدام نفس البرنامج الذي تعرفنا عليه في المثال السابق ثم نقوم بالإضافات المطلوبه لأدخال ملف فيديو للمثال.

1- في الخطوة الأولى (بعد قراءتك للأمثلة الماضية يفترض أن تكون هذه الخطوة الآن بديهية لديك) نقوم بتعريف الكائنين الجديدين : كائن الفيديو وكائن الصوت (تذكر ان كل ملف فيديو يجب ان يحتوي على صوت حتى لو كان صوت غير

مسموع , بمعنى ان كل ملف فيديو يجب ان يحتوي على قناة صوتية سبق وحفظت معه). وذلك في ملف التعاريف Main.h كما يلي :

```
DECLARE AGDXAvi* Avi;
DECLARE AGDXSound* Soun
```

```

Main.h
#ifndef MAIN_H
#define MAIN_H
#include <windows.h>
#include <windowsx.h>
#include <AGDX.h>
#ifdef Ex1_CPP
#define DECLARE
#else
#define DECLARE extern
#endif
DECLARE HWND hWnd;
DECLARE BOOL bActive;
// The AGDXScreen object, every program must have one
DECLARE AGDXScreen* Screen;
// Sound objects
DECLARE AGDXMusic* Music;
// Background objects
DECLARE AGDXLayer* sky;
DECLARE AGDXLayer* ground;
DECLARE AGDXLayer* mountain1;
DECLARE AGDXLayer* mountain2;
// OK lets now look at thd Avi and Sound objects
DECLARE AGDXAvi* Avi;
DECLARE AGDXSound* Sound;
// our program runs on different computer speeds
// so we need this to control the speed of our program
DECLARE int diffTick,firstTick,lastTick,WaitTime;
void Start_Program(void);
void Finish_Program(void);
void Main(void);
#endif

```

2- بعد ذلك في الخطوة الثانية ننتقل الى وظيفة البداية ونقوم بما يلي :

- أولاً تغيير عدد الألوان المستخدمة من 256 لون (أو يعرف بـ 8 بت) الى 65536 لون (أو يعرف بـ 16 بت) عن طريق تغيير الرقم من 8 الى 16 في أمر انشاء كائن الشاشة :

```
Screen->CreateFullScreen(hWnd, 640, 480, 16);
```

- ثانياً ننشئ كائن الصوت :

```
Sound = new AGDXSound;
Sound->Create(hWnd);
```

- الآن نستطيع ان ننشئ كائن الفيديو كما يلي :

```
if(!Avi->Create(Screen, Sound, "intro.avi", FALSE))
}
    MessageBox(hWnd, "Avi Create", "Error", MB_OK);
{
```

الملف الذي سوف نستخدمه هو "intro.avi" وقد مررناه الى كائن الفيديو مع كائن الشاشة وكائن الصوت. طبعاً اضفنا الأمر الشرطي "if" لكي نتأكد أن العملية مرت بنجاح وتم قبول الملف.

- بعد أن انشأنا كائن الفيديو نستطيع ان نقوم باستخدامه، وهذا هو المكان المناسب لإعطائه الأمر بتشغيل الملف السينمائي وذلك كما يلي :

```
if(!Avi->Start())
}
    MessageBox(hWnd, "Avi Start", "Error", MB_OK);
{
```

نقوم هنا بالتأكد أن العملية مرت بنجاح , لاحظ اننا لا نحتاج للتأكد ولكن من حسن العادة في البرمجة حساب الأخطاء المستقبلية فإذا حصل خطأ في البرنامج نعرف ما هو السبب وأين حصل.

- أخيرا في هذه الوظيفة نقوم عن طريق كائن الفيديو لأمره بجعل حجم ملف الفيديو بحجم الشاشة بأكملها (لاننا في هذا المثال نستخدم البعد 640 x 480 وحجم ملف الفيديو المستخدم هو 320 x 240) وذلك عن طريق الأمر التالي :

```
Avi->SetDest(0, 0, 480, 640);
```

3- الخطوة الثالثة هي الوظيفة الرئيسية وما نريده هو ان يشتغل الملف حال تشغيل البرنامج ثم يختفي بعد ذلك من الشاشة ويبدأ البرنامج في عملة. لكي نقوم بذلك نستخدم الفعل الشرطي " **if** " بحيث نجعل الكمبيوتر يراقب الملف فإذا وصل لنهايته (اي اذا تعادل المتغير `Avi->m_Index` الدال على عدد الأطارات المستخدمة مع المتغير `Avi->m_nFrames` الدال على مجموع الأطارات) عندها نبدأ في تنفيذ البرنامج.

4- الخطوة الرابعة والأخيرة في وظيفة النهاية واعتقد انها سهلة الفهم فكل ما نقوم به هو إيقاف الملف الموسيقي عن اللعب وكذلك تحرير الذاكرة التي سبق وحجزناها للكائنات.

بقية البرنامج سبق وشرحناه في المثال السابق "الخلفيات الطبقيه".

ونرى الملف الرئيسي Main.cpp والشامل على جميع الوظائف السابقة كاملا كما يلي :

```
#include "main.h"

// This is our main program
// http://www.ArabGames.com
// Because we are using video we don't forget to link to
// (vfw32.lib) plus of course (winmm.lib dinput.lib dsound.lib
// ddraw.lib agdx.lib dxguid.lib and the other default libs.)
// Look in the top menu (Project -> Settings.. -> Link)

void Start_Program(void)
{
    bActive=TRUE;
    // Create the AGDXScreen object and set the resolution
    Screen = new AGDXScreen();
    // Set the Screen resolution and number of colors
    Screen->CreateFullScreen(hWnd, 640, 480, 16);
    // Load the palette from the files bitmap
    Screen->LoadPalette("sky.bmp");
    // lets first load all the layers
    sky = new AGDXLayer(Screen,"sky.bmp");
    ground = new AGDXLayer(Screen,"ground.bmp");
    mountain1 = new AGDXLayer(Screen,"m1.bmp");
    mountain2 = new AGDXLayer(Screen,"m.bmp");
    // lets set the Key Color
    mountain1->ColorKey(0);
    mountain2->ColorKey(0);
    //Creat the AGDXMusic object and use the window handler
    Music = new AGDXMusic(hWnd);
    Music->Play("furelise.mid");
    // Create the sound object
    Sound = new AGDXSound;
    Sound->Create(hWnd);
    // lets take care of the avi file now
    Avi = new AGDXAvi();
    if(!Avi->Create(Screen, Sound, "intro.avi", FALSE))
    {
        MessageBox(hWnd, "Avi Create", "Error", MB_OK);
    }
    if(!Avi->Start())
    {
        MessageBox(hWnd, "Avi Start", "Error", MB_OK);
    }
}
```

```
Avi->SetDest(0, 0, 480, 640);
}

void Finish_Program(void)
{
// we are finished so lets Delete all the objects and free the memory
Music->Stop();
SAFE_DELETE(Avi);
SAFE_DELETE(Sound);
SAFE_DELETE(sky);
SAFE_DELETE(ground);
SAFE_DELETE(mountain1);
SAFE_DELETE(mountain2);
SAFE_DELETE(Music);
SAFE_DELETE(Screen);
// now we can just exit the program
PostQuitMessage (0);
}

void Main(void)
{
// ----to control the time-----
firstTick = timeGetTime();
diffTick = firstTick - lastTick;
lastTick = firstTick;

WaitTime += diffTick;
// -----
if (Avi->m_Index == Avi->m_nFrames)
{
if (WaitTime > (20) )
{
mountain1 ->ScrollRight(2);
mountain2 ->ScrollRight(3);
ground ->ScrollRight(5);
WaitTime =0;
}
sky ->Draw(0,0,Screen->GetBack());
mountain1 ->DrawTrans(0,190,Screen->GetBack());
mountain2 ->DrawTrans(0,200,Screen->GetBack());
ground ->Draw(0,317,Screen->GetBack());
} else Avi->Draw(Screen->GetBack());
//Ok, now flip the Screen
Screen->Flip();
}
```

المثال الثالث عشر

المثال الثالث عشر الطبقات شبه الشفافة وملفات برنامج PhotoShop



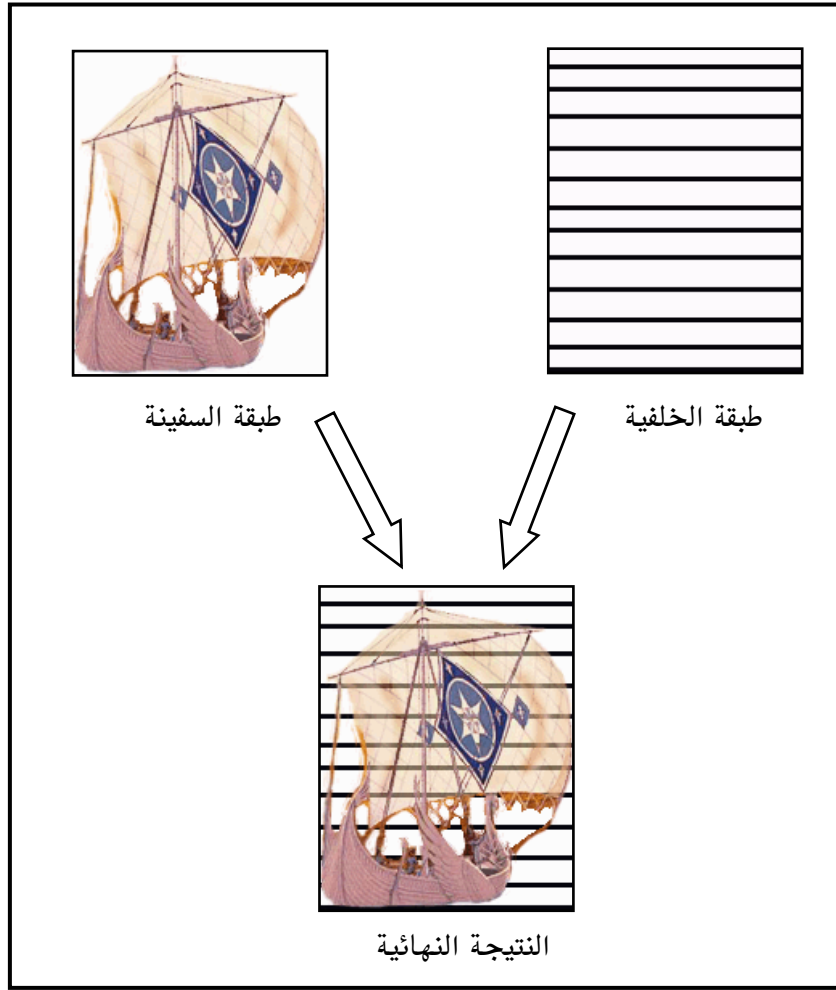
نستطيع عن طريق المكتبة AGDX أن نتعامل مع الطبقات شبه الشفافة. وهي الطبقات التي نستطيع أن نرى من خلالها الخلفية ولكننا نراها في نفس الوقت. ونستطيع استخدام ملفات البرنامج الشهير **PhotoShop** في رسم تلك الطبقات ثم استخدامها في برامجنا. ويتمتع برنامج **PhotoShop** وملفاته التي تنتهي بالأحرف PSD بخاصية الطبقيّة. بمعنى أننا نستطيع أن نقسم الصورة إلى طبقات "Layers" وكل طبقة تحيط بها منطقة شفافة. المتمرسين طبعاً على هذا البرنامج يعرفون ما نعني بالطبقات. وتستطيع مكتبتنا أن تستغل جميع الطبقات الموجودة في ملف الصورة الناتجة عن هذا البرنامج. وأود أن أوضح بأن نوعية الملفات لهذا البرنامج نستطيع إنتاجها كذلك عن طريق البرنامج **PaintShop** والذي يعادل ثمنه مائة دولار فقط مقارنةً بستمائة دولار للبرنامج الأول.

في المكتبة AGDX سوف نقوم بإنشاء طبقة لـ DirectX ثم نقوم بوضع ملف الصورة في تلك الطبقة. بعد ذلك سنعرضها على الشاشة بدون أن تفقد ميزة القناة شبه الشفافة (والتي تعرف كذلك بقناة ألفا "Alpha") وبدون أن نفقد أي من الطبقات الموجودة في تلك الصورة. كذلك نستعرض ميزة أخرى جديدة في هذا المثال قد تكون ممتازة لأولئك الذين يحبون برمجة ألعاب المغامرات. فمن العوامل المعروفة بصعوبتها عند إنشاء الخلفيات هي كيفية جعل الممثلين يقفون أمام أو خلف شيء معين في الشاشة. سنقوم نحن بهذه الميزة وبدون استخدام أي أوامر برمجية بل إن كل ما في الأمر أننا سنقوم برسم صورة إضافية نطلق عليها "Depth Mask" أو قناع العمق والتي عن طريقها سنحدد العمق لكل جزء من الصورة الأصلية. نعم أكاد أسمعك تقول كيف؟ ماذا تقصد؟ عن ماذا نتحدث؟ لا عليك سوف نحاول عن طريق هذا المثال القيام بهذه العملية وبالشرح الممل.

أولاً افرض أننا نريد تصوير مشهد تمر به سفينة أمام جزيرة. ونريد من أشعة تلك السفينة أن تكون شبه شفافة. بمعنى أننا لا نريدها أن تكون شفافة كما فعلنا من قبل مع "مفاتيح الألوان" ولكننا نريدها أن تكون شبه شفافة بحيث نستطيع أن نراها ونرى المنطقة التي تغطيها.

سنقوم أولاً باستخدام برنامج PhotoShop أو برنامج PaintShop أو أي برنامج يستطيع التعامل مع ملفات الصور من نوع PSD برسم السفينة كما نشاء. بعد ذلك سنقوم بإعطاء درجة الشفافية للأشعة عن طريق البرامج السابقة. بعدها نقوم بحفظها في ملف على القرص المصلب (مثلاً في هذا المثال اسم الملف ship.psd).

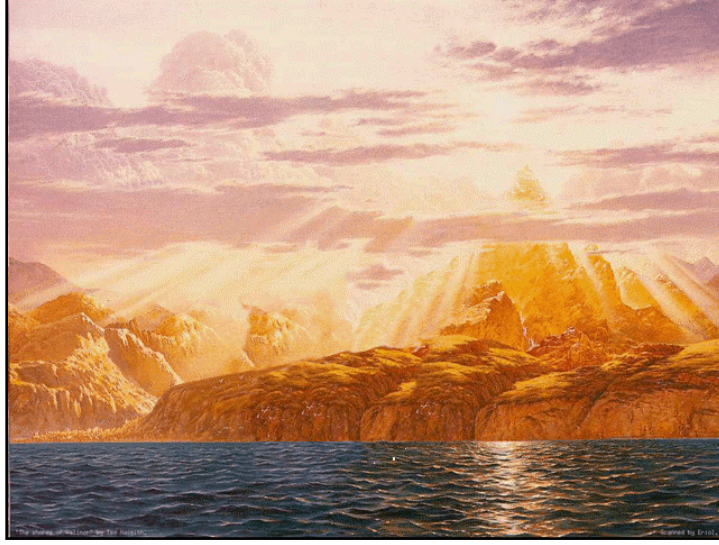
انظر للشكل في الصفحة المقابلة لتأخذ فكرة أوضح عن ما نتحدث.



كما تلاحظ في الشكل الأعلى ، النتيجة النهائية للسفينة عندما نضعها على الخلفية نستطيع أن نرى الأشرطة شبه الشفافة. طبعا الخلفية في المثال ستكون عبارة عن شاطئ البحر أمام جزيرة. ومهما كانت صورة الخلفية فأن النتيجة ستكون واحدة.

يجب أن نستخدم ملايين الألوان (أو 16 بت) عند التعامل مع هذه النوعية من الصور.

الآن لنعود لموضوع ميزة "قناع العمق" والتي تحدثنا عنها في بداية هذا المثال. ونلقي نضر للخلفية في هذا المثال ونراها كما يلي :

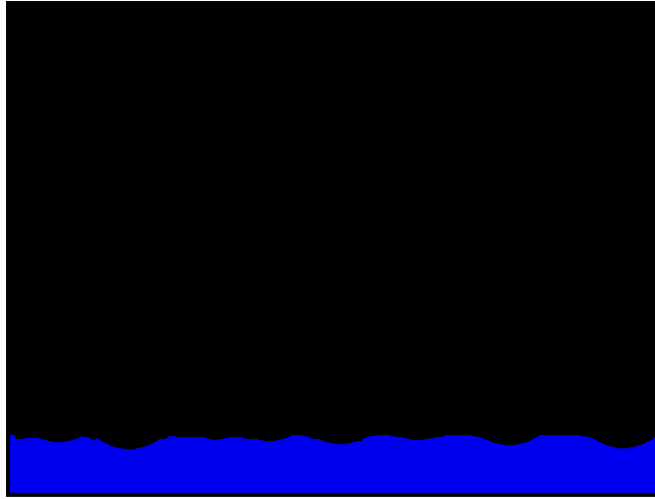


وافرض مثلاً أننا نود أن نعطي بعض العمق للبحر ، بحيث إذا مرت السفينة عليه فإن جزء من المياه يكون خلفها وجزء آخر يكون أمامها. وكما أوضحنا أننا نستطيع القيام بذلك بدون أي شفرة برمجية وسنقوم باستخدام أحد البرامج السابقة لرسم الصورة والتي سنستخدمها "كقناع للعمق".

ولكن قبل أن نتكلم عن كيفية رسم قناع العمق يجب أن نفهم ما نعني بالـ "عمق". العمق هو المسافة بين اقرب مدى لك وهي شاشة الكمبيوتر وأبعد مدى لك وهو ابعاد مكان في داخل صورة الخلفية. أو نستطيع أن نطلق عليه البعد الثالث (سنتعرف عليه لاحقاً في المكتبة Genesis 3D). وسنقسم هذا العمق إلى 256 درجة بحيث يكون الرقم 0 هو ابعاد عمق ممكن والرقم 256 هو اقرب عمق ممكن.

ولكن ألم نقل بأننا لن نستخدم أي شفرة برمجية لإنتاج العمق ؟ نعم لذلك سنعتبر عن تلك الأرقام باللون الأزرق بحيث يكون اللون الأسود هو عبارة عن الرقم صفر واللون الأزرق هو عبارة عن الرقم 255. ونستخدم اختلاف الحدة بينهما للتعبير عن العمق. أو للنظر لها من زاوية أخرى نحن نعلم أن اللون الأزرق يعبر عنه بالرقم $(0,0,255)$ أي خليط الألوان احمر Red واخضر Green وأزرق Blue. وبما أننا نريد فقط اللون الأزرق فكلما اللونين الآخرين يحملان القيمة صفر. ونحن نعلم أن اللون الأسود يعبر عنه بالقيمة $(0,0,0)$ وكما ترى أن قيمة اللون الأزرق هي التي تغيرت إلى الصفر ونستطيع أن نصف اختلاف الشدة بينهما بتغيير الرقم من 0 إلى 255. ويسمح لك برنامج الرسم بالقيام بذلك.

فندرى مثلاً الصورة التي استخدمناها في هذا المثال كما يلي :



طبعاً يجب أن يكون بنفس حجم الخلفية وميزة قناع العمق ليست شرطاً لاستخدام الطبقات شبه الشفافة. فنستطيع أن نهملها ونستخدم فقط الطبقات الشفافة إذا أردنا ولكن بما أنها ميزة مهمة قد نستخدمها في برامجك فقد فضلنا ذكرها.

الآن لندخل إلى القسم البرمجي لهذا البرنامج. أولاً ليس هناك اختلاف في الكثير من أجزاء هذا المثال عما سبق وشرحناه في الأمثلة الماضية، وسنقوم بنفس العملية لإنشاء الكائنات المطلوبة وكيفية استخدامها.

ملف التعاريف Main.H :

سوف نرى كيف نقوم أولاً بتعريف الكائنات الجديدة في هذا المثال :

```
DECLARE AGDXMusic* Music;
DECLARE AGDXInput Input;
DECLARE AGDXLayer* Background;
DECLARE AGDXSurface* Ship;
```

لاحظ أننا لم نقم بأي تغيير عن الأمثلة السابقة واحتجنا إلى تعريف وإنشاء بعض الكائنات ، كائن خاص بالملفات الموسيقية "AGDXMusic" وكائن آخر مسؤول عن الإدخال في البرنامج (نستطيع استخدام هذا الكائن للحصول على الإدخال من إشارة الفأرة أو عصا الألعاب أو لوحة المفاتيح) "AGDXInput". بعد ذلك قمنا بتعريف كائن الطبقات لاستخدامه للخلفية "AGDXLayer". ثم عرفنا كائن خاص بطبقات DirectX لاستخدامه في التعامل مع ملفات البرنامج **PhotoShop** ولحفظ صورة السفينة بالتحديد.

بقية ملف التعاريف مطابقة للأمثلة السابقة.

الملف الرئيسي Main.cpp ، وظيفة البداية (Start_Program):

أما في وظيفة البداية فنقوم بما يلي :

```
bActive=TRUE;

// Create the AGDXScreen object and set the resolution
Screen = new AGDXScreen();

// Set the Screen resolution and nubmer of colors
Screen->CreateFullScreen(hWnd, 640, 480, 16);

Background = new AGDXLayer(Screen, "Back.bmp");
Background->Draw(Screen->GetBack());

Music = new AGDXMusic(hWnd);
Music->Play("SC.mid");

Ship = new AGDXSurface();
Ship->CreatePSD(Screen, "ship.psd", 157, 201, true, "depths.psd");
Ship->DrawPSD(0, 235, Screen->m_lpDDSBack);

Screen->Flip();
```

وفيها قمنا بإنشاء الكائنات كما سبق وشرحنا في الأمثلة السابقة والجديد هنا هو كائن أسطح DirectX والذي نعرفه بـ AGDXSurface. عن طريقة قمنا بإنشاء سطح

في الذاكرة لتعبئة صورة السفينة. بعد ذلك استخدمنا الأمر :

```
Ship->CreatePSD(Screen,"ship.psd",157,201,true,"depths.psd");
```

والذي يعني أننا نريد تعبئة الملف ship.psd في ذلك السطح ، والأرقام 157,201 تعني طول وعرض الصورة ، والأمر true يعني أننا نريد استخدام ميزة الطبقات الشفافة ، وأخيرة تعبئة الملف " depths.psd " والحامل لقناع العمق. وكما سبق وذكرنا أن هذا الملف اختياري فيمكن إرفاقه أو إهماله.

بعد ذلك نرى الأمر التالي :

```
Ship->DrawPSD(0,235,Screen->m_lpDDSBack);
```

ويعني أننا نريد أن نرسم صورة السفينة على السطح الخلفي في الموقع $x=0$ و $y=235$. وسنستخدم هذا الأمر في الوظيفة الرئيسية لاحقا للقيام بنفس العملية مع تغيير موقع السفينة.

الملف الرئيسي Main.cpp ، الوظيفة الرئيسية Main(void):

الآن دعونا نرى كيف يكون شكل الوظيفة الرئيسية:

```
void Main(void)
{

static int movex=0,movey=235;

if( movex < 640) movex+=2; else movex = -60;

Background->Draw(Screen->GetBack());
Ship->DrawPSD(movex,movey,Screen->GetBack());

Screen->Flip();

}
```

بكل بساطة:

- نقوم بإنشاء وإعطاء المتغيرات المستخدمة قيمها الأولية (لاحظ استخدام الكلمة **static** وتعني إعطاء القيم لهذا المتغير مرة واحدة طوال عمل البرنامج).
 - بما أننا نريد من السفينة أن تتحرك على المحور x فأنا نقوم بتغيير المتغير movex بمعدل نقطتين لكل وحدة زمنية فإذا ما وصلت قيمته للرقم 640 أي آخر الشاشة من اليمين نعطيه القيمة -60 أي أول الشاشة من اليسار و هلم جرا.
- بعد ذلك نقوم برسم المطلوب على السطح الخلفي وقلبه للسطح الأمامي.

بالنسبة لوظيفة النهاية فهي كما تعودنا أن نحرر الكائنات التي سبق
واستخدمناها. ونرى الملف الرئيسي للبرنامج كما يلي :

```
#include "main.h"

void Start_Program(void)
{
    bActive=TRUE;

    // Create the AGDXScreen object and set the resolution
    Screen = new AGDXScreen();

    // Set the Screen resolution and nubmer of colors
    Screen->CreateFullScreen(hWnd, 640, 480, 16);

    Background = new AGDXLayer(Screen, "Back.bmp");
    Background->Draw(Screen->GetBack());

    Music = new AGDXMusic(hWnd);
    Music->Play("SC.mid");

    Ship = new AGDXSurface();
    Ship->CreatePSD(Screen,"ship.psd",157,201,true,"depths.psd");
    Ship->DrawPSD(0,235,Screen->m_lpDDSBack);

    Screen->Flip();
}
```

```
void Finish_Program(void)
{
    SAFE_DELETE(Ship);
    SAFE_DELETE(Music);
    SAFE_DELETE(Background);
    SAFE_DELETE(Screen);

    PostQuitMessage (0);
}

void Main(void)
{
    static int movex=0,movey=235;

    if( movex < 640) movex+=2; else movex = -60;

    Background->Draw(Screen->GetBack());
    Ship->DrawPSD(movex,movey,Screen->GetBack());

    Screen->Flip();
}
```

ونرى ملف التعاريف كما يلي :

```
#ifndef MAIN_H
#define MAIN_H
#include <AGDX.h>

#ifdef Ex1_CPP
#define DECLARE
#else
#define DECLARE extern
#endif
```

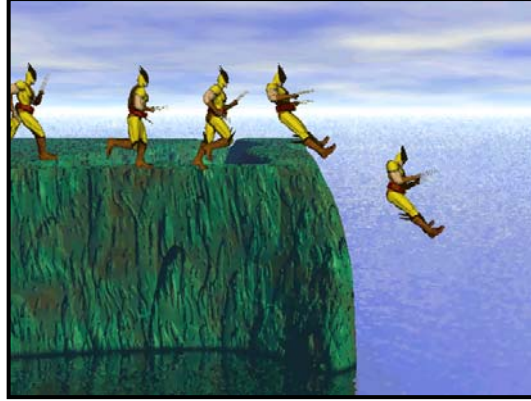
```
DECLARE AGDXScreen* Screen;
DECLARE HWND hWnd;
DECLARE BOOL bActive;

// Sound objects
DECLARE AGDXMusic* Music;
DECLARE AGDXInput Input;
DECLARE AGDXLayer* Background;
DECLARE AGDXSurface* Ship;

void Start_Program(void);
void Finish_Program(void);
void Main(void);
```

المثال الرابع عشر

المثال الرابع عشر
نوع آخر من الممثلين وطرق الإدخال
لوحة المفاتيح - إشارة الفأرة - عصا الألعاب



تعرفنا فيما سبق من الأمثلة على الممثلين. وقد سبق وذكرت أن في هذه المكتبة نوعين من الممثلين. وبما أننا تحدثنا عن النوع الأول في الأمثلة السابقة ، فإننا في هذا المثال سنتكلم عن النوع الثاني.

ولكن لماذا نستخدم نوعين ؟ لأننا في النوع الأول قمنا باستخدام الكائن `AGDXSprite` وهو الكائن الخاص والمسؤول عن الممثلين. وهذا الكائن يقوم بالتعامل مع نسخة واحدة للمثل. بمعنى أننا نستطيع استخدامه مع أكثر من ممثل في المشهد ولكننا سنحتاج لإنشائه عن طريق الأمر `new` وإزالته عن طريق الأمر `SAFE_DELETE()` مع كل ممثل على حدا. ولكن ماذا لو أردنا أن نكرر ممثل واحد مرات عديدة في المشهد ؟ طبعا استخدام كائن الممثل لن يجدي في هذه الحالة لان هذا يعني أننا سوف نقوم بإنشاء العدد المطلوب من الممثلين ثم تحريرهم من الذاكرة كل ما استغينا عنهم. وتكرار هذه العملية بكثرة يؤدي ولا شك لمشاكل في الذاكرة ناهيك عن البطء لهذه العملية.

افرض مثلا انك تطلق الرصاص في مشهد معين. فأنت تعرف أن شكل الرصاصة وحركتها متطابق مع كل الرصاص الخارج من ذلك المسدس. ولكنك تريد أن تستخدم عدد من الممثلين للتعبير عن تلك الطلقات كلما أردت أن تطلق النار. طبعاً بعد أن تختفي الطلقات من الشاشة لا تريدها أن تبقى في الذاكرة وهنا نرى فائدة هذا النوع من الممثلين. حيث نستطيع تكرار عدد مرات ظهور الممثل على الشاشة عن طريق إضافته وإزالته في أي وقت نشاء بدون الحاجة لاستخدام الأمر `new` أو الأمر `(SAFE_DELETE)` كلما أردنا إضافة أو إزالة نسخة من ذلك الممثل.

ولكن كيف استطعنا القيام بهذه العملية بدون استخدام الأمرين `new` و `(SAFE_DELETE)` ؟ استطعنا عن طريق استخدام ما يسمى بالقائمة المتصلة (الفصل الحادي عشر في الجزء الثاني من هذا الكتاب مخصص لشرح معنى وكيفية إنشاء القائمة المتصلة. راجعة إذا أحببت معلومات أكثر عنها) والتي سمحت لنا القيام بهذه العملية وبشكل بسيط كما سوف نرى في هذا المثال.

في هذا المثال سنقوم باستخدام بطلنا في الجري على حافة الهاوية (كما ترى في الصورة في بداية هذا المثال) ثم القفز إلى الماء. كما سنقوم بتكرار هذا الممثل كلما ضغطنا على الزر الأيسر لإشارة الفأرة أو على الزر الأيسر (أو الرئيسي) لعصا الألعاب أو مفتاح المسافة على لوحة المفاتيح. أيضاً في هذا المثال نريد أن نوضح لك سهولة استخدام أدوات الإدخال المتوفرة على الجهاز.

قبل الدخول في شرح هذا المثال ، أريد أن أوضح كيف أنني استطعت تصميم وتحريك الممثل وكذلك تصميم الخلفية :

- قمت أولاً في برنامج Poser بإنشاء الممثل وهناك العديد من الأشكال الجاهزة التي تستطيع اختيارها وتغييرها حسب رغبتك (هذا البرنامج مشروح في الفصل التاسع بشكل مفصل).
- بعد ذلك وباستخدام البرنامج السابق قمت بحفظ الحركات المطلوبة (الجرى والقفز في هذا المثال) في صور متعددة على القرص الصلب في جهازي.



hero.bmp

- بعد ذلك استخدمت البرنامج Tyler (مرفق على القرص المدمج الملحق بالكتاب) لجمع جميع تلك الصور وبالترتيب المطلوب في صورة واحدة (hero.bmp). يجب أن تتأكد من أن عرض تلك الصورة لا يزيد عن عرض الشاشة وهي 640 نقطة في هذا المثال. وهذا البرنامج يسمح لك بالتحكم في ذلك.
- ثم تستطيع أن تستخدم أي برنامج رسم (مثلاً أنا استخدم برنامج Photoshop لتعديل وتنقيح تلك الصور حسب الحاجة ويستحسن تحويلها من ملايين الألوان إلى 256 لون)
- وأخيراً في هذه العملية قمت باستخدام البرنامج Bryce 3D (والمشروح في الفصل الثامن بشكل مفصل) لإنشاء الخلفية للمشهد.

هذه هو كل ما قمت به لإنشاء الصور والخلفية المطلوبة لهذا المثال ، واستغرقت العملية كلها ساعة واحدة فقط.

لندخل الآن للقسم البرمجي لهذا البرنامج. أولاً ليس هناك أي اختلاف في الكثير من أجزاء هذا المثال عما سبق وشرحناه في الأمثلة السابقة. وسنقوم بنفس العملية لإنشاء الكائنات المطلوبة واستخدامها.

ملف التعاريف Main.H :

سوف نرى كيف نقوم أولاً بتعريف الكائنات في هذا المثال :

```
// Sound objects
DECLARE AGDXMusic* Music;
DECLARE AGDXInput Input;
DECLARE AGDXLayer* Background;
DECLARE RECT Window;
DECLARE AGDXTile* hero;
DECLARE AGDXSpriteList Sprites;

// Sprite types
enum
{
    SPR_hero,
};

DECLARE int diffTick,firstTick,lastTick,WaitTime;
```

لاحظ أننا لم نقم بأي تغيير عن الأمثلة السابقة واحتجنا إلى تعريف وإنشاء بعض الكائنات ، كائن خاص بالملفات الموسيقية " AGDXMusic " وكائن آخر مسؤول عن

الإدخال في البرنامج (نستطيع استخدام هذا الكائن للحصول على الإدخال من إشارة الفأرة أو عصا الألعاب أو لوحة المفاتيح) "AGDXInput". ثم بعد ذلك قمنا بتعريف كائن الطبقات لاستخدامه للخلفية "AGDXLayer". ثم عرفنا متغير من نوع RECT وهو Window ويعبر عن حدود المستطيل الذي سنعرض فيه الممثلين (في هذا المثال فأن المستطيل يشمل الشاشة بأكملها) ونستخدم هذا المتغير إذا أردنا تقسيم الشاشة ، مثلا قسم لمعلومات عن المشهد وقسم للصور والحركة. بعد ذلك أنشأنا كائن خلفيات التايلز AGDXTile والذي تعرفنا عليه فيما سبق ولكن هنا لن نستخدمه لخلفيات التايلز وإنما لهذا النوع من الممثلين وسنستخدمه للتعبير عن الممثل hero. ثم نقوم بإنشاء كائن القائمة المتصلة AGDXSpriteList. وأخيرا سنحتاج لإنشاء متغير من نوع enum كما يلي :

```
enum
{
    SPR_hero,
};
```

وإذا أردنا إضافة ممثل جديد نقوم بإضافة اسم معين (ليس مهم ما هو الاسم) في هذه القائمة ولكن من المهم وجود اسم تعريف لكل ممثل مستخدم. وسترى السبب في شرح الوظيفة SpriteHero(void) فيما بعد. بقية ملف التعاريف مطابقة للأمثلة السابقة.

الملف الرئيسي Main.cpp ، وظيفة البداية (Start_Program):

أما في وظيفة البداية فنقوم بما يلي :

```
bActive=TRUE;

// Create the AGDXScreen object and set the resolution
Screen = new AGDXScreen();

// Set the Screen resolution and nubmer of colors
Screen->CreateFullScreen(hWnd, 640, 480, 16);

Background = new AGDXLayer(Screen, "Back.bmp");

Music = new AGDXMusic(hWnd);
Music->Play("Didi.mid");

hero = new AGDXTile(Screen,"hero.bmp", 136, 136, 27);
hero->ColorKey(0);

SetRect(&Window, 0, 0, 640, 480);
```

وفيها قمنا بإنشاء الكائنات كما سبق وشرحنا في الأمثلة السابقة. والجديد هنا هو طريقة استخدام كائن التابلز AGDXTile ، ففي هذا المثال يعبر عن ممثل معين.

واستخدمناه كما يلي :

```
hero = new AGDXTile(Screen,"hero.bmp", 136, 136, 27);
hero->ColorKey(0);
```

قمنا أولاً بإعطاء الصورة التي يوجد بها إطارات الحركة لهذا الممثل (وهي كما سبق وشرحنا في هذا المثال توجد في الصورة hero.bmp) بعد ذلك نعطي طول وعرض كل إطار للحركة في الصورة. ثم عدد مجموع الإطارات 27 في هذه الحالة. وأخيراً نحدد نوع اللون المفتاح وهو اللون الأسود أو القيمة 0.

الملف الرئيسي Main.cpp ، الوظيفة الرئيسية Main(void):

الآن دعونا نرى كيف يكون شكل الوظيفة الرئيسية:

```
firstTick = timeGetTime();
diffTick = firstTick - lastTick;
lastTick = firstTick;
WaitTime += diffTick;
```

أولاً في أعلى هذه الوظيفة استخدمنا هذه الأوامر لنحصل على الوقت الحقيقي وقد سبق وقمت بذلك في بعض الأمثلة السابقة. ونقوم بهذه العملية لكي نستطيع أن نتحكم بشكل أدق في سرعة البرنامج. لأن قوة المعالجات تختلف من جهاز لآخر فلا نريد الحركة أن تكون سريعة على جهاز وبطيئة على آخر. واستخدمت الأمر timeGetTime() للحصول على الوقت الحقيقي. وباستخدام عدد من المتغيرات من نوع int استطعنا استغلال ميزة الوقت الحقيقي في المثال.

بعد ذلك نرى ما يلي :

```
static int Count = 0;

Input.Update();

if(Input.Keys[DIK_SPACE] || Input.MouseLB || Input.JoystickB1)
{
    if(--Count < 0)
    {
        Sprites.AddSprite(hero, SPR_hero,-136,75);
        Count = 50;
    }
}

if(!Input.Keys[DIK_SPACE] && !Input.MouseLB && !Input.JoystickB1)
Count = 0;
```

في هذا الجزء استخدمنا كائن الإدخال للحصول على المعلومات المطلوبة عن إشارة الفأرة أو عصا الألعاب أو لوحة المفاتيح. وعن طريق استخدام الأمر الشرطي (if) قمنا بإضافة نسخة من الممثل على الشاشة عن طريق الأمر :

```
Sprites.AddSprite(hero, SPR_hero,-136,75);
```

وكما ترى استخدمنا الكائن hero ثم استخدمنا اسم التعريف للممثل SPR_hero وكل ممثل -كما سبق وأوضحنا- يجب أن يملك اسم تعريف والذي عرفناه في ملف التعريف سابقا. بعد ذلك أعطينا قيم الموقع (السيني والصادي) على الشاشة والتي نود ظهور الممثل بها.

وكما تلاحظ أننا استخدمنا كائن الإدخال `Input` لنعرف هل مفتاح المسافة تم ضغطه وكذلك الزر الأيسر لإشارة الفأرة أو الزر الرئيسي لعصا الألعاب. وتُرجع أما صحيح أم غير صحيح.

بقية الأوامر وضعت لكي نتحكم في سرعة إضافة الممثل.

في الجزء الأخير من هذه الوظيفة نرى التالي:

```

if (WaitTime > (40) )
{
    WaitTime =0;
    SpriteHero();
}

Background->Draw(Screen->GetBack());
Sprites.DrawClipped(Screen->GetBack(), &Window);
Screen->Flip();

}

```

وكما نرى هنا أننا استخدمنا المتغيرات التي أنشأناها للوقت للتحكم في عدد مرات استدعاء الوظيفة `SpriteHero()` والتي تشمل سر اختلاف هذا النوع من الممثلين عن النوع الأول. وسوف ننظر إليها بالتفصيل بعد قليل.

ثم بعد ذلك نرى المعتاد وهو رسم الصور المطلوبة للسطح الخلفي. لاحظ أننا استخدمنا الأمر:

```
Sprites.DrawClipped(Screen->GetBack(), &Window);
```

لرسم الممثل للسطح الخلفي ولاحظ كذلك كيف استخدمنا المتغير `Window`. بعد ذلك قلب السطح الخلفي للسطح الرئيسي عن طريق الأمر `Screen->Flip()`.

الوظيفة SpriteHero(void)

الآن حان الوقت للنظر للوظيفة SpriteHero(void) فهي الوظيفة المختلفة في هذا المثال (انظر للصفحة التالية) وكل ما سبق رأيناه في الأمثلة السابقة. بما أننا في هذه النوعية من الممثلين سنقوم بإنشاء عدد غير محدد لكل ممثل من هذا النوع ، فإننا نريد من كل نسخة من ذلك الممثل أن تكون مستقلة عن النسخ الأخرى. وهذا ما تقوم به هذه الوظيفة. ولا نحتاج لوظيفة مختلفة لكل ممثل لأننا نستطيع أن نستخدم نفس الوظيفة لكل الممثلين في وقت واحد للمشاهد. وكل ما نقوم به عندما يكون لدينا أكثر من ممثل هو استخدام اسم التعريف (تذكر عندما شرحنا ملف التعريف وكذلك عندما استخدمنا أمر الإضافة في الوظيفة الرئيسية ذكرنا بأن كل ممثل يحتاج لأسم تعريف) مع الأمر الشرطي (switch) كما يلي :

```
switch(Node->m_Type)
{
case SPR_hero:
{
هنا تضع الشفرة البرمجية لهذا الممثل
} break;
case : ممثل آخر :
{
هنا تضع الشفرة البرمجية لهذا الممثل
} break;
}
```

وبما أننا نستخدم—كما سبق وذكرنا كذلك—القائمة المتصلة فيجب أن نعلم بأن

```
////////////////////////////////////  
// UpdateHero  
////////////////////////////////////  
void SpriteHero(void)  
{  
    AGDXSprite* Node;  
    AGDXSprite* Save;  
  
    for(Node = Sprites.Next(); Node != Sprites.List(); Node = Save)  
    {  
        Save = Node->m_Next;  
  
        switch(Node->m_Type)  
        {  
            case SPR_hero:  
            {  
                Node->m_PosX +=6;  
  
                if(Node->m_PosX < 230)  
                {  
                    if(Node->m_Frame == 14) Node->m_Frame=0;  
                }  
  
                Node->SetFrame(Node->m_Frame);  
                if(Node->m_Frame < 26) Node->m_Frame++;  
                else Node->m_PosY +=10;  
  
                if(Node->m_PosY > 480 || Node->m_PosX > 640)  
                {  
                    Sprites.DelSprite(Node);  
                }  
  
            } break;  
        }  
    }  
}
```

القائمة المتصلة باختصار هي عبارة مجموعة نقاط اتصال Node وكل نقطة اتصال تمثل نسخة من الممثل. ونقوم بالتعامل مع كل نقطة من تلك النقاط على أساس أنها الممثل ونستطيع إضافة أو حذف أي نقطة نشاء في أي وقت (هذا لإجابة السائلين عن كيفية التعامل مع الذاكرة بإضافة وحذف الممثلين خلال عمل البرنامج). طبعاً إذا كنت تريد معلومات إضافية عن القوائم المتصلة فأنصحك بمراجعة الفصل الحادي عشر في الجزء الثاني من هذا الكتاب.

الآن عرفنا أننا نتعامل مع كائن اسمه Node عن طريقة نستطيع التحكم في الموقع ، الإطار ، السرعة ، التسارع والكثير من المواصفات التي نحتاجها للممثلين.

عند بداية هذه الوظيفة نقوم بحفظ نقطة الاتصال التي سنتعامل معها (لأننا نتعامل مع نسخة واحدة أو نقطة واحدة لممثل معين كل مرة نأتي بها لهذه الوظيفة) ونقوم بهذه العملية كما يلي :

```
AGDXSprite* Node;
AGDXSprite* Save;

for(Node = Sprites.Next(); Node != Sprites.List(); Node = Save)
{
    Save = Node->m_Next;
}
```

بعد أن نقوم بهذه العملية الروتينية نبدأ باستخدام الأمر الشرطي (switch) في النظر لكل الممثلين الموجودين في المشهد. لأننا نستطيع استخدام وظيفة واحدة فقط للتحكم في كل الممثلين. وبما أننا لم نستخدم في هذا المثال إلا ممثل واحد " hero " فإننا سنقوم باستخدام اسم التعريف التابع له " SPR_hero " للتحكم فيه.

ونرى هذه العملية كما يلي :

```
switch(Node->m_Type)
{
case SPR_hero:
{
Node->m_PosX +=6;

if(Node->m_PosX < 230)
{
if(Node->m_Frame == 14) Node->m_Frame=0;
}

Node->SetFrame(Node->m_Frame);
if(Node->m_Frame < 26) Node->m_Frame++;
else Node->m_PosY +=10;

if(Node->m_PosY > 480 || Node->m_PosX > 640)
{
Sprites.DelSprite(Node);
}

} break;
```

ونرى كيف نتحكم في موقع الممثل وحركته. هنا مثلاً قمنا بتغيير الموقع السيني للممثل `m_PosX` والإطارات من 0 إلى 14 (هذه إطارات المشي في الصورة `hero.bmp`) إلى أن يصل للنقطة 230 ثم يتغير لحالة القفز ونقوم بتغيير الموقع الصادي له `m_PosY` والإطارات من 15 إلى 26 (هذه إطارات القفز). وأخيراً عندما يصل الممثل إلى أي موقع سيني `m_PosX` أكبر من 640 أي نهاية الشاشة من اليمين أو إلى أي موقع صادي `m_PosY` أكبر من 480 أي نهاية الشاشة من الأسفل فأننا نقوم بمسحة من قائمة الاتصال وبتالي من الذاكرة عن طريق الأمر التالي :

```
Sprites.DelSprite(Node);
```

الملف الرئيسي Main.cpp ، وظيفة النهاية Finish_Program():

```
void Finish_Program(void)
{
    int maxsprite = Sprites.m_nSprites;

    for (int delcnt=0; delcnt < maxsprite; delcnt++)
    {
        Sprites.DelSprite(Sprites.Next());
    }

    SAFE_DELETE(hero);
    SAFE_DELETE(Music);
    SAFE_DELETE(Background);
    SAFE_DELETE(Screen);

    PostQuitMessage (0);
}
```

وظيفة النهاية في هذا المثال تختلف قليلا عن الأمثلة السابقة. فهنا قمنا بمسح جميع نقاط القائمة المتصلة من الذاكرة والتي قد تكون موجودة في القائمة لسبب أو لآخر.

أولا: حصلنا على عدد النقاط الموجودة في القائمة ووضعناها في المتغير maxsprite بعد ذلك تأكدنا من استخدام أمر المسح (والذي سبق شرحه في الوظيفة السابقة) :

```
Sprites.DelSprite(Sprites.Next());
```

وذلك مع كل نقطة حتى نتأكد أن القائمة أصبحت خالية من أي نقاط. أما بقية الوظيفة فهو مطابق لما سبق وشرحناه. ونقوم طبعاً بإزالة بقية الكائنات من الذاكرة.

الشفرة البرمجية للمثال الرابع عشر

الملف Main.cpp

```
#include "main.h"

void Start_Program(void)
{
    bActive=TRUE;

    // Create the AGDXScreen object and set the resoultion
    Screen = new AGDXScreen();

    // Set the Screen resolution and nubmer of colors
    Screen->CreateFullScreen(hWnd, 640, 480, 16);

    Background = new AGDXLayer(Screen, "Back.bmp");

    Music = new AGDXMusic(hWnd);
    Music->Play("Didi.mid");

    hero = new AGDXTile(Screen,"hero.bmp", 136, 136, 27);
    hero->ColorKey(0);
    SetRect(&Window, 0, 0, 640, 480);
}

void Finish_Program(void)
{
    int maxsprite = Sprites.m_nSprites;

    for (int delcnt=0; delcnt < maxsprite; delcnt++)
    {
        Sprites.DelSprite(Sprites.Next());
    }
}
```

```
SAFE_DELETE(hero);
SAFE_DELETE(Music);
SAFE_DELETE(Background);
SAFE_DELETE(Screen);

        PostQuitMessage (0);
}

void Main(void)
{
// ----to control the time-----
    firstTick = timeGetTime();

    diffTick = firstTick - lastTick;
    lastTick = firstTick;

    WaitTime += diffTick;
// -----

    static int Count = 0;

    Input.Update();

    if(Input.Keys[DIK_SPACE] || Input.MouseLB || Input.JoystickB1)

    {
        if(--Count < 0)
        {
            Sprites.AddSprite(hero, SPR_hero,-136,75);
            Count = 50;
        }
    }

    if(!Input.Keys[DIK_SPACE] && !Input.MouseLB && !Input.JoystickB1)
    Count = 0;

    if (WaitTime > (40) )
    {
        WaitTime =0;
        SpriteHero();
    }
}
```

```
Background->Draw(Screen->GetBack());

// Loop the list and draw the sprites
Sprites.DrawClipped(Screen->GetBack(), &Window);

Screen->Flip();
}

/////////////////////////////////////////////////////////////////
// UpdateHero
/////////////////////////////////////////////////////////////////
void SpriteHero(void)
{
    AGDXSprite* Node;
    AGDXSprite* Save;

    // Loop the list and update the sprites
    for(Node = Sprites.Next(); Node != Sprites.List(); Node = Save)
    {
        Save = Node->m_Next;

        switch(Node->m_Type)
        {
            case SPR_hero:
            {
                Node->m_PosX +=6;

                if(Node->m_PosX < 230)
                {
                    if(Node->m_Frame == 14) Node->m_Frame=0;
                }

                Node->SetFrame(Node->m_Frame);

                if(Node->m_Frame < 26) Node->m_Frame++;
                else Node->m_PosY +=10;

                if(Node->m_PosY > 480 || Node->m_PosX > 640)
                {
                    Sprites.DelSprite(Node);
                }
            }
        }
    }
}
```



```
        }  
    } break;  
}  
}
```

الملف Main.h

```
#ifndef MAIN_H  
#define MAIN_H  
  
#include <windows.h>  
#include <windowsx.h>  
#include <AGDX.h>  
  
#ifdef Ex1_CPP  
#define DECLARE  
#else  
#define DECLARE extern  
#endif  
  
DECLARE AGDXScreen* Screen;  
DECLARE HWND hWnd;  
DECLARE BOOL bActive;  
  
// Sound objects  
DECLARE AGDXMusic* Music;  
DECLARE AGDXInput Input;  
DECLARE AGDXLayer* Background;  
DECLARE RECT Window;  
DECLARE AGDXTile* hero;
```

```
DECLARE AGDXSpriteList Sprites;

// Sprite types
enum
{
    SPR_hero,
};

// our program runs on different computer speeds
// so we need this to control the speed of our program
DECLARE int diffTick,firstTick,lastTick,WaitTime;

void Start_Program(void);
void Finish_Program(void);
void Main(void);
void SpriteHero(void);

#endi
```


المثال الخامس عشر

المثال الخامس عشر نريد أن نبرمج برنامج كاملاً ونبيعه في الأسواق



حتى هذه النقطة كل الأمثلة السابقة استخدمت لعرض ميزة أو أكثر من مميزات المكتبة AGDX. واستخدمنا مشهد واحد للتعبير عن ذلك. ولا بد أنك الآن توجه لنفسك هذا السؤال: " هذا جيد ، ولكن كيف أستطيع جمع كل تلك المميزات (أو المشاهد) في برنامج واحد ؟ لكي أنتج برنامج كامل صالح للتوزيع ". إذا كنت قد فهمت جميع الأمثلة السابقة فأن هذا المثال سيكون كذلك سهل الفهم لديك. في هذا المثال سنحاول أن نكتب شفرة برمجية تستطيع أن تجعلها بداية لبرنامجك. كما تستطيع أن تستخدمها بأي شكل تريده عن طريق إضافة الوظائف والمميزات التي تريدها ، لتصبح برنامج كامل صالح للتوزيع. وكما تعلم أننا في هذا الكتاب لم نستخدم البرمجة الكائنية (استخدمناها فقط في التركيب الداخلي للمكتبة AGDX) واستخدمنا البرمجة الوظيفية بدل عنها، أي أننا اعتمدنا أكثر على عمل الوظائف وذلك لجعل الكتاب أكثر سهوله للفهم وخاصة للذين لم يسبق لهم

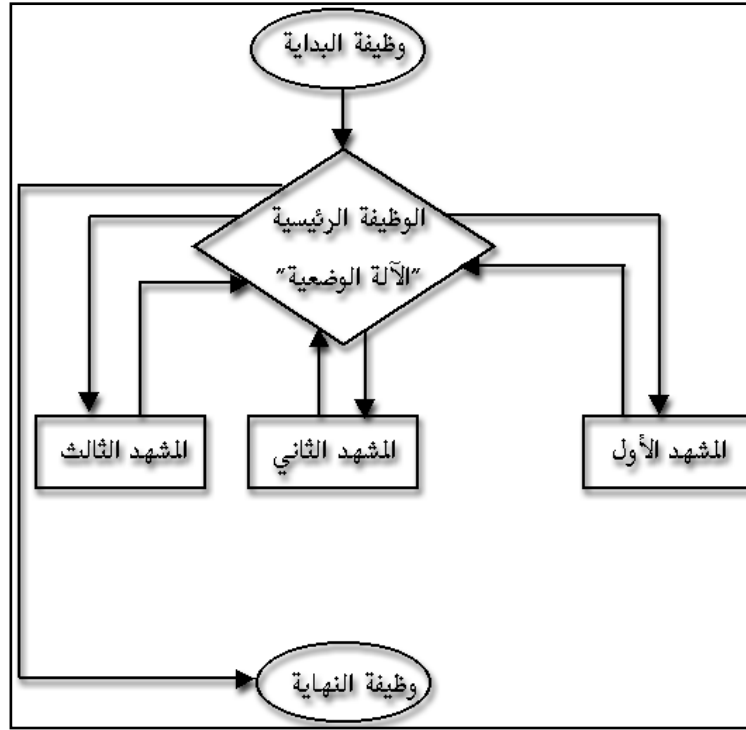
البرمجة على أنظمة النوافذ من قبل. فإننا في هذا المثال سنشرح طريقة للتحكم بشكل كامل (والكمال لله) في عمل تلك الوظائف ، ولكن ليس على مستوى الوظيفة الواحدة بل على مستوى البرنامج ككل وسنستخدم ميكانيكية تعرف بـ "آلة الـ State machine".

الآلة الـ الوضعية

ما هي الآلة الـ الوضعية ؟ الآلة الـ الوضعية ليست سفينة فضاء ، وليست آله زمنية تعيدنا للماضي و هي كذلك ليست آله جيب إلكترونية صغيرة. في الحقيقة الآلة الـ الوضعية ليست آلة بمعنى الكلمة ولكنها طريقة أو آلية برمجية سنتعرف عليها لتعطينا تحكم كامل في عمل البرنامج وجمع جميع المشاهد والمميزات التي نريدها فيه .

عندما نبرمج أي برنامج ، فإننا على علم مسبق بالوظائف الرئيسية التي ستكون به. ونحن كذلك على علم مسبق بعلاقتها وكيفية تعاملها مع بعضها البعض. وعندما يكون البرنامج ضخماً فإن من المستحسن أن نرسم رسم بياني يوضح تلك العلاقة في البرنامج. مثلاً لهذا المثال سنقوم برسم بياني بسيط (في الصفحة المقابلة). وفيه نرى بأن الوظيفة الرئيسية هي التي تحدد أي مشهد نذهب إليه ثم نعود منه أو ننهي البرنامج. ونحن نعلم أن كل مشهد قد يحتوي على كل أو بعض المميزات التي نريد إلحاقها ، من ممثلين وأصوات وأفلام فيديو وحركة... الخ. ونحن نعلم

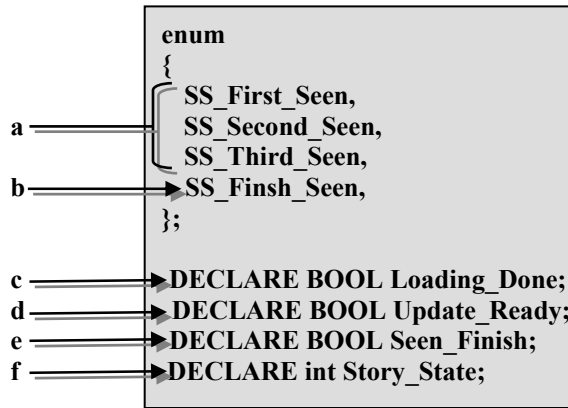
بأن كل مشهد يجب أن يحرر نفسه من الذاكرة عند الانتهاء من عملة ويجب أن يحرر جميع الكائنات التي تم إنشاؤها فيه.



رسم بياني لعلاقة الوظائف في البرنامج وطريقة عمل الآلة الوضعية

ولكن كيف استطاعت الوظيفة الرئيسية الحصول على كل هذا التحكم؟ طبعاً باستخدام الآلة الوضعية. ونقوم بإنشاء هذه الآلة عن طريق إنشاء عدة متغيرات تكون أجزاء لها. وكما تعودنا في الأمثلة السابقة، سننشئ أي متغيرات لبرامجنا

في الملف `main.h`. ولذلك سنقوم بإنشاء المتغيرات المطلوبة فيه وفي هذا المثال قمنا بإنشاء ما يلي من المتغيرات المطلوبة للآلة الوضعية:



لأننا سنقوم بإنشاء ثلاث مشاهد (طبعا في برنامجك تستطيع أن تنشئ مئات المشاهد) فإننا سنحتاج إلى ثلاث متغيرات للتعبير عنها (a). وكذلك سنحتاج لمتغير للتعبير عن حالة إنهاء البرنامج (b). وكل مشهد كما سوف نرى يحتوي على جزء لإنشاء وتعبئة الكائنات في الذاكرة (c) ، وجزء ثاني للقيام بالتجديد المطلوب للمشهد (d) ، وجزء ثالث لتحرير الذاكرة عند الانتهاء منها (e). وأخير سنحتاج إلى متغير للتعبير عن الآلة الوضعية نفسها (f).

لاحظ أن المتغيرات (a) و (b) هي من نوعية `enum` وهذه النوعية تحمل قيم ترتيبها ابتداء من الصفر إلى قيمة ترتيب آخر عنصر بها. في هذه الحالة مثلاً المتغير `SS_Finsh_Seen` يحمل القيمة "ثلاثة" (راجع الصفحة 23 في فصل الأساسيات لمعلومات أكثر عن هذا النوع من المتغيرات).

أما المتغيرات (c) و (d) و (e) فأنها من نوع `Bool` وهي تحمل قيمتين : أما صحيح أو غير صحيح. وأخيراً المتغير (f) والذي يعبر عن الآلة الوضعية هو من

نوع `int` لأن المتغيرات (a) و (b) والتي تعبر عن حالة البرنامج هي في الحقيقة أرقام من نوع `int`.

ربما يكون الشرح السابق للآلة الوضعية دسماً شيئاً ما ولكن خلاصة الحديث أن كل المتغيرات التي استخدمناها هي عبارة عن `enum` و `bool` و `int`. ونحتاجها للتعبير عن كل حالة من حالات البرنامج (المشهد الأول، المشهد الثاني، نهاية البرنامج، ...) وتستطيع إضافة حالات أخرى في برامجك الخاصة بإضافة مشاهد جديدة ووظائف جديدة لكل مشهد وبنفس الطريقة.

بعد ذلك في الملف `main.h` نقوم بما سبق ورأيناه في جميع الأمثلة السابقة وهو تعريف الكائنات التي سنستخدمها في برنامجنا. وكذلك تعريف جميع الوظائف التي سنستخدمها. وفي هذا المثال نحن استخدمنا ثلاث وظائف إضافية للتعبير عن المشاهد الثلاث للبرنامج :

```
void First_Seen(void);
void Second_Seen(void);
void Third_Seen(void);
```

طبعا تستطيع إضافة ما تشاء من الوظائف لاستخدامها في برامجك. ولجعل الأمور أكثر سهولة سنكتب كلاً منها في ملف `cpp` خاص بها كما سنرى ذلك لاحقاً.

بعد أن قمنا بتعريف كل ما سنستخدمه في البرنامج، نستطيع أن نبدأ كتابة الشفرة البرمجية لمثالنا. وحتى تكون المسألة سهلة فأننا سنقوم أولاً بما تعودنا عليه وهو أن نبدأ من وظيفة البداية في الملف الرئيسي `main.cpp`.

الملف الرئيسي Main.cpp ، وظيفته البداية (Start_Program):

أما في وظيفته البداية فنقوم بما يلي :

```
void Start_Program(void)
{
    bActive=TRUE;

    Loading_Done = false;
    Update_Ready = false;
    Seen_Finish = false;
    Story_State = SS_First_Seen;

    // Create the AGDXScreen object and set the resolution
    Screen = new AGDXScreen();

    // Set the Screen resolution and nubmer of colors
    Screen->CreateFullScreen(hWnd, 640, 480, 8);

}
```

وكما ترى أنها لا تحتوي على الكثير من الأوامر. فقد قمنا في بدايتها بإعطاء

المتغيرات الضرورية للآلة الوضعية قيمها الأولية :

Loading_Done هذا المتغير هو المسؤول عن تعبئة الكائنات في المشاهد وهو الجزء

المطابق لوظيفة البداية لكل مشهد. وأعطيناها القيمة غير صحيح

"false" لأخبار الآلة الوضعية أن هذا المشهد لم يعبئ في الذاكرة

بعد.

Update_Ready هذا المتغير هو المسؤول عن الحركة الرئيسية في المشاهد وهو الجزء المطابق للوظيفة الرئيسية لكل مشهد. وأعطيناها القيمة غير صحيح "false" لأخبار الآلة الوضعية أن هذا المشهد لم يبدأ عملة بعد.

Seen_Finish هذا المتغير هو المسؤول عن تحرير الكائنات التي استُخدمت في المشهد من الذاكرة وهو الجزء المطابق لوظيفة النهاية لكل مشهد. وأعطيناها القيمة غير صحيح "false" لأخبار الآلة الوضعية أن هذا المشهد لم يتم بحجز أي ذاكرة لتحريرها بعد.

Story_State هذا المتغير هو المسؤول عن وضع الآلة الوضعية ونقوم عن طريق تغيير هذا المتغير إلى توجيه الكمبيوتر للقيام بالعمل المطلوب فعلة في البرنامج. في هذه الحالة وبما أننا في وظيفة البداية أعطيناها القيمة -المشهد الأول- "SS_First_Seen" لأخبار الآلة الوضعية أننا نريد من البرنامج التوجه إلى المشهد الأول.

بعد ذلك قمنا بإنشاء الكائن الرئيسي للمكتبة AGDX وطلبنا منه استغلال الشاشة بكاملها واستخدام "256" لون (لماذا 256 لون فقط؟ الإجابة بعد قليل).
 إذاً أين هي أوامر إعداد الكائنات؟ لأننا قسمنا برنامجنا لأكثر من مشهد (سابقاً استخدمنا الملف الرئيسي main.cpp ليحتوي على مشهد واحد متكامل) ولأننا فصلنا كل مشهد بملف خاص به. ونحن نعلم بأن كل مشهد سيحتاج لممثلين وأصوات خاصة به. فان من الحكمة جعل كل مشهد يهتم بإنشاء وتحرير الكائنات بنفسه ، بدل وضعها هنا في الملف الرئيسي.

الملف الرئيسي Main.cpp ، الوظيفة الرئيسية Main(void):

الآن دعونا نرى كيف يكون شكل الوظيفة الرئيسية:

```
void Main(void)
{
// ----to control the time-----
    firstTick = timeGetTime();
    diffTick = firstTick - lastTick;
    lastTick = firstTick;
    WaitTime += diffTick;
// -----

switch(Story_State)
{
    case (SS_First_Seen):
        {
            First_Seen();
        } break;

    case (SS_Second_Seen):
        {
            Second_Seen();
        } break;

    case (SS_Third_Seen):
        {
            Third_Seen();
        } break;

    case (SS_Finsh_Seen):
        {
            Finish_Program();
        } break;

}
}
```

وكما نرى بأن لولا الأوامر المتعلقة بالتوقيت الحقيقي والتي سبق وشرحناها في أمثلة سابقة فإن الوظيفة الرئيسية هي قلب الآلة الوضعية. واستخدمنا الأمر الشرطي :

```
switch(Story_State)
```

ليختبر حالة المتغير Story_State وعلى حسب قيمته يتوجه البرنامج للمكان المطلوب. وهو أما "المشهد الأول" أو "المشهد الثاني" أو "المشهد الثالث" أو "وظيفة النهاية".

وهذه هي "الآلة الوضعية" أو State machine.

بالنسبة لوظيفة النهاية فهي كما تعودنا أن نحرر الكائنات التي سبق واستخدمناها. ونراها كما يلي :

```
void Finish_Program(void)
{
    SAFE_DELETE(Screen);
    PostQuitMessage (0);
}
```

وكما ترى أنها صغيرة للغاية لأن كل ما تقوم به هو تحرير الكائن الرئيسي للمكتبة AGDX والخروج من البرنامج.

هل نستخدم 256 لون فقط أم آلاف لألوان

٢

في اغلب الأمثلة السابقة لم نتطرق لمسألة الألوان ، واستخدمنا آلاف الألوان أو ما يعرف بـ 16 BIT وابتعدنا عن مسائل حصر الألوان لكل مشهد ولم نهتم في عدد الألوان التي استخدمناها أو ما هي. في هذا المثال سنقوم باختيار 256 لون أو ما يعرف بـ 8 BIT. لماذا ؟ لأن هذا العدد من الألوان يكفي لكل مشهد ومحتوياته. وقد يظن القارئ أننا عندما نتكلم عن هذا العدد من الألوان بأننا نقصد استخدام نفس ملف الألوان في كل البرنامج. ولكن المكتبة AGDX تسمح لنا باستخدام ملف ألوان مختلف لكل مشهد ، مما يجعل هذا العدد من الألوان محلاً للاهتمام وذلك باستخدام هذه الميزة. صحيح أن الكثير من بطاقات العرض أصبحت ذات قدرات عالية على عرض ملايين الألوان ولكن لازال استخدام 256 لون لكل مشهد يوفر علينا الكثير من ذاكرة العرض ، كما يسمح بإضافة خواص حركية كثيرة للمشهد. وهناك مجموعة من الأوامر الجذابة في هذه المكتبة والتي تعمل فقط على هذا العدد من الألوان. طبعاً تحويل هذا المثال لاستخدام آلاف الألوان مسألة بسيطة للغاية وقد اطلعنا عليها في الكثير من الأمثلة السابقة. وكل ما نحتاج فعله للقيام بذلك هو إزالة الأوامر البرمجية التي تعمل فقط مع 256 لون (إذا لم نزلها لن يحدث شي ، كل ما في الأمر أنها لن تعمل ولكن البرنامج سيعمل). وتغيير الرقم 8 إلى 16 عند إنشاء الكائن الرئيسي لهذه المكتبة في وظيفة البداية.

بما أننا في هذا المثال سنستخدم 256 لون فهذا يعني أننا سنتعامل مع لوحة الألوان ولهذا سوف نقوم بإنشاء متغير خاص بها في ملف التعاريف كما يلي :

```
DECLARE PALETTEENTRY pe[256];
```

وسنقوم باستخدام هذا المتغير لتغيير ملف الألوان للشاشة وذلك لكل مشهد. كما أننا سنستخدمه مع بعض الأوامر لإعطاء بعض المؤثرات.

	المشهد الأول
--	---------------------

في هذا المثال سنقوم بشرح المشهد الأول فقط. لأن باقي المشاهد مشابه له وبالتالي ستكون مفهومة. وكما سبق وأوضحنا أن الملف الرئيسي مكون من ثلاث وظائف :

❖ وظيفة البداية

❖ ووظيفة النهاية

❖ والوظيفة الرئيسية

وهذا التقسيم مشابه جدا لتقسيم المشهد لأننا سنحتاج للقيام بنفس العملية في كل مشهد (إنشاء الكائنات ، استخدامها ، مسحها من الذاكرة) وكل مشهد هو عبارة عن وظيفة واحدة سبق وعرفناها في ملف التعاريف. وسوف نقوم بتقسيم هذه الوظيفة إلى ثلاث أقسام مشابه لتقسيم الملف الرئيسي. وسوف نستخدم المتغيرات الثلاث " Loading_Done ، Update_Ready ، Seen_Finish " والتي سبق وذكرناها

لتحكم في كل قسم من أقسام الوظيفة. نحن نعلم أن هذه المتغيرات تحمل قيمة صحيح (true) للدخول إلى ذلك القسم أو غير صحيح (false) لعدم الدخول إليه. لماذا قمنا بذلك؟ لأننا نحتاج في قسم من الوظيفة لإنشاء الكائنات التي سنستخدمها في المشهد. تماما كما كنا نفعل مع وظيفة البداية في الملف الرئيسي. وسنستخدم المتغير Loading_Done في معرفة ما إذا قمنا بإنشاء الكائنات بعد أم لا. وكذلك سنحتاج إلى قسم نقوم فيه بالعمليات الرئيسية في المشهد. تماما كما كنا نفعل في الوظيفة الرئيسية في الملف الرئيسي وسنستخدم المتغير Update_Ready لمعرفة هل أصبح المشهد جاهزاً ليقوم بتلك العمليات أم لا، وهنا نقوم بالدخول لهذا القسم مباشرة بعد الانتهاء من القسم السابق.

أخيراً سنحتاج إلى قسم نقوم فيه بتحرير الكائنات من الذاكرة تماما كما كنا نفعل مع وظيفة النهاية في الملف الرئيسي. ونستخدم المتغير Seen_Finish لمعرفة إذا كان الوقت صحيحاً لإنهاء المشهد. سنقوم في قسم العمليات الرئيسية بوضع هذا المتغير مساوياً لـ "صحيح" عندما ننتهي من المشهد وذلك لكي تذهب الآلة الوضعية لهذا القسم فتبدأ عملية التحرير.

قسم الإنشاء :

```

if (!Loading_Done)
{

// Load the palette from the bitmap
Screen->LoadPalette("seen1.bmp");
Screen->GetPalette(0,256,pe); // Save the palette in the pe variable

Background = new AGDXLayer(Screen, "seen1.bmp");

```

```

Background->Draw(Screen->GetBack());

Screen->FillPalette(0,0,0);
Screen->Flip(); // Flip the back buffer to the front
Screen->FadeIn(6,pe); // fade in the pe palette

eyes=new AGDXSprite(Screen,"eyes.bmp", 22, 9, 2);
eyes->m_PosX = 417;
eyes->m_PosY = 283;

Music = new AGDXMusic(hWnd);
Music->Play("qalby.mid");

// Create the sound object
Sound = new AGDXSound();
Sound->Create(hWnd);

SND_Rain = new AGDXSoundBuffer();
SND_Rain->Load(Sound,"rain.wav", 1);
SND_Rain->Play();

Loading_Done = true;
Update_Ready = true;
}

```

وكما نرى أننا أنشأنا المتغيرات بالإضافة للأوامر الضرورية عن بدأ المشهد. ونشرحها هنا بالترتيب. أولاً قمنا بتعبئة ملف الألوان للمنظر الخلفي للمشهد وتخزينه في المتغير `pe` كما يلي:

```

Screen->LoadPalette("seen1.bmp"); // تعبئة ملف الألوان من المنظر الخلفي
Screen->GetPalette(0,256,pe); // حفظ ملف الألوان في متغير ملف الألوان

```

لأننا عند بداية ونهاية كل مشهد سنقوم بتغيير تدريجي من وإلى اللون الأسود ثم إلى خلفية المشهد. (بهذه الطريقة لا يستطيع المستعمل أن يرى تغير ملف الألوان لأن التغيير سوف يحصل عندما تكون الشاشة سوداء ، وتستخدم الكثير من البرامج في الأسواق هذه الطريقة لتغيير ملف الألوان). كما يلي :

```
Screen->FillPalette(0,0,0); ←————— تعبئة ملف الألوان باللون الأسود
Screen->Flip(); ←————— قلب السطح الخلفي للسطح الرئيسي
Screen->FadeIn(6,pe); ←————— الآن نقوم بتعبئة ملف الألوان بشكل تدريجي للشاشة
```

ونقوم بهذه العملية في كل مشهد ، حتى نرى خلفية الشاشة تظهر تدريجيا من اللون الأسود.

وأخيرا نرى ما يلي :

```
Loading_Done = true;
```

```
Update_Ready = true;
```

وهنا نخبر الآلة الوضعية بأننا انتهينا من تعبئة المشهد وحن الوقت للدخول للقسم الرئيسي له .

(بقية شفرة قسم الإنشاء سبق وشرحناها مرات كثيرة في الأمثلة السابقة)

قسم العمليات الرئيسية :

```

if(Update_Ready)
{

    if (WaitTime > (2000) )
    {

        /// this is great mothed for looping 0 to 2 then back 2 to 0
        static int Frame=0;
        static int Delta = -1;
        if(Frame == 0 || Frame == 1) Delta = -Delta;

        eyes->SetFrame(Frame+=Delta);

        if (Frame == 0) WaitTime =0; else WaitTime =1900;

    };

    if(!SND_Rain->Play()) SND_Rain->Play();

    eyes->Draw(Screen->GetFront());

    Input.Update();
    if(Input.Keys[DIK_SPACE]) Seen_Finish= true;

}

```

في هذا القسم قمنا بكتابة كل العمليات الرئيسية للمشهد وإذا كنت متابعاً للأمثلة السابقة فسترى أنها جميعاً مألوفة لديك. وكل ما قمنا به ، هو تحريك ممثل "أعين الحيوان الصغير" على حسب التوقيت الحقيقي. ويقوم هذا القسم بمراقبة مفتاح المسافة من على لوحة المفاتيح. فإذا حصل وضُغِط يقوم بإعطاء المتغير Seen_Finish القيمة "صحيح" وذلك لنخبر الآلة الوضعية بأن الوقت قد حان لإنهاء المشهد والذهاب للقسم الأخير لهذه الوظيفة.

قسم تحرير الكائنات:

```

if(Seen_Finish)
{
    Seen_Finish = false;
    Loading_Done = false;
    Update_Ready = false;

    Screen->FadeOut(0);

    Music->Stop();
    SND_Rain->Stop();

    SAFE_DELETE(SND_Rain);
    SAFE_DELETE(Sound);
    SAFE_DELETE(Music);
    SAFE_DELETE(eyes);
    SAFE_DELETE(Background);

    //now we are sure we will go to the next seen
    Story_State = SS_Second_Seen;
}

```

بعد أن أنهينا المشهد نقوم أولاً بوضع المتغيرات الثلاث " Loading_Done ، Update_Ready ، Seen_Finish " لوضعها الأولي قبل الدخول في المشهد وهو "غير صحيح" وذلك استعداداً لبدأ مشهد جديد أو لإنهاء البرنامج.

ثم استخدمنا الأمر التالي :

```
Screen->FadeOut(0);
```

والذي يقوم بتحويل ألوان الشاشة بالتدريج من الصورة الخلفية للمشهد إلى اللون الأسود (لاحظ أن جميع الأوامر التي تتعامل مع ملف الألوان تعمل فقط في بيئة

256 لون ، راجع مرجع المكتبة AGDX في آخر الكتاب لمعلومات إضافية عن كل أم.

ثم قمنا كالمعتاد بتحرير الكائنات. وأخيرا في هذا القسم قمنا بتوجيه الآلة الوضعية للمشهد الثاني في البرنامج عن طريق الأمر التالي :

```
Story_State = SS_Second_Seen;
```

وكما ترى أننا عن طريق هذا الأمر نوجه الآلة الوضعية إلى حيث نشاء. وهنا وجهناها إلى المشهد الثاني للبرنامج ، ثم في المشهد الثاني سنوجهها إلى المشهد الثالث أو نستطيع الرجوع إلى المشهد الأول. باختصار هذا المتغير يعطينا تحكم كامل في طريقة سير البرنامج. ثم إننا في المشهد الأخير سوف نوجه الآلة الوضعية إلى إنهاء البرنامج عن طريق التوجه إلى وظيفة النهاية في الملف الرئيسي للبرنامج. وبما أننا في هذا القسم قمنا بتحرير جميع الكائنات في هذا المشهد فإن كل ما سنحتاجه في الوظيفة الرئيسية هو تحرير الكائن الرئيسي للمكتبة AGDX وإنهاء البرنامج كما سبق وأوضحنا عند شرحها.

هل هذه هي كل مميزات مكتبة الألعاب ؟

لا ، هناك الكثير من المميزات الأخرى التي لم نتطرق إليها، مثل تشغيل الأغاني من الأقراص المدمجة في البرامج التي تستخدمها ، وإمكانية صنع برامج اتصال عبر الإنترنت ، وإمكانية تخزين صورة الشاشة على ملف من نوع bmp ، وإمكانية استخدام كل GUI التابع لـ win32 (مثل مربعات التحديث والقوائم العلوية للنوافذ وغيرها). كذلك إمكانية التعرف على ميكانيكية التصادم. ولكن للأسف هذا هو

آخر مثال مشروح لهذه المكتبة في هذا الكتاب. وكما ترى أن الكتاب اخذ الكثير من الوقت والجهد ، وحجمه ليس بالصغير ولم يكن بالإمكان إضافة شرح أي أمثلة إضافية. كما أنني لم أرى للآن أي كتاب آخر في الأسواق يأتي مع هذا الكم الهائل من الأمثلة. ومع هذا كله فإن هناك الكثير من الأمثلة الإضافية الموجودة في الملف (Extra) على القرص المدمج التابع للكتاب والتي تستطيع أن تتعلم منها أشياء إضافية. وبما أن المكتبة AGDX تطورت من المكتبة CDX والشكر طبعاً للأستاذ Danny Farley والذي لم يمانع في استخدام مكتبته بالشكل الذي رأيتته مناسب مع هذا الكتاب ، لذلك عملت على تحويل جميع أمثلة تلك المكتبة لتعمل مع المكتبة AGDX وذلك لزيادة الفائدة المرجوة (كما أن هناك لعبة كاملة على القرص المدمج مع الشفرة البرمجية تستطيع الإطلاع عليها).

بوضع هذا في الحسبان اعتقد أن الكتاب أدى واجبه بالشكل الوافي وتم شرح هذه المكتبة للمرحلة التي تستطيع فيها عزيزي القارئ الاعتماد على نفسك في استعمالها. كما أن هناك مرجع لكل أوامرها (ماعدات التي تم إضافتها مؤخراً وهي قليلة) موجود في الفصل الأخير من هذا الكتاب.

ليس هدف الكتاب الحقيقي هو استعمال هذه المكتبة فقط ولكن معرفة كيفية التعامل مع تكنولوجيا DirectX ، لذلك ستجد الشفرة البرمجية كاملة لهذه المكتبة ويمكنك تغييرها كما تشاء حسب متطلباتك.

Special Thanks to Danny Farley (for making his wonderful CDX library free, and for allowing me to use its code in this book. AGDX is built on top of CDX 1.5 and has been made especially for this book only. Many other thanks for every one whom helped with CDX as well.

Unfortunately Danny is not continuing CDX development anymore, however; a group of talented people has taken over the project. Thanks to them too.

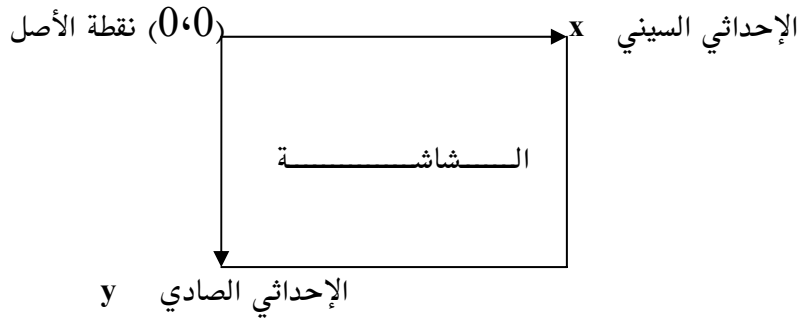
الفصل السابع

الحركة المفصلة

والذكاء الاصطناعي

الحركة المفصلة والذكاء الاصطناعي

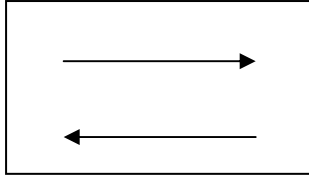
من السهل جداً تحريك موقع نقطة معينة على الشاشة (قد تكون هذه النقطة موقع ممثل مثلاً) بشكل مستقيم ، فكل ما علينا فعله هو تغيير الإحداثي السيني x أو الإحداثي الصادي y أو كلاهما لتحريك موقع تلك النقطة في اتجاه معين. وذلك باستخدام الشاشة لتحديد موقع أي نقطة، ونحن نعلم أن الزاوية في أعلى يسار الشاشة هي نقطة الأصل (أو تعرف كذلك بنقطة الصفر) وليس في المنتصف. وفي بداية برامجنا نقوم بتحديد عدد النقاط الظاهرة على الشاشة في كلا الاتجاهين السيني x أو الإحداثي الصادي y . مثلاً في حالة لو أننا استخدمنا الصفاء 640×480 فهذا يعني أن هناك 480 نقطة ظاهرة على الإحداثي السيني تبدأ من نقطة الأصل (الزاوية في أعلى اليسار من الشاشة) وكذلك 640 نقطة على الإحداثي الصادي تبدأ كذلك من نقطة الأصل. لذلك نستطيع تحديد عدد النقاط الظاهرة على الشاشة عن طريق تحديد البعد المطلوب.



مثلا لتغيير موقع ممثل معين على الإحداثي السيني x فإننا نستطيع استخدام المعادلة البسيطة هذه:

تغيير الموقع من اليسار لليمين بمعدل نقطة واحدة $x = x + 1$;
أو

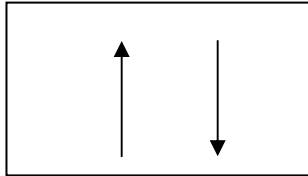
تغيير الموقع من اليمين للييسار بمعدل نقطة واحدة $x = x - 1$;



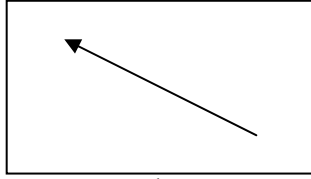
وتنطبق القاعدة على الإحداثي الصادي y كما يلي :

تغيير الموقع من الأعلى للأسفل بمعدل نقطة واحدة $y = y + 1$;
أو

تغيير الموقع من الأسفل للأعلى بمعدل نقطة واحدة $y = y - 1$;

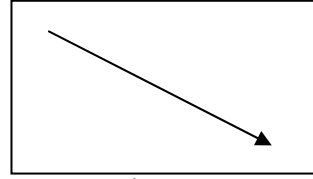


وكذلك نستطيع أن نغير كلا الإحداثيين x و y في آن واحد كما يلي :



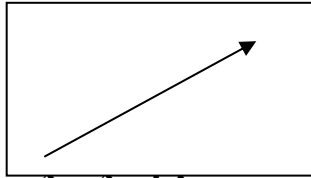
$$x = x - 1 ;$$

$$y = y - 1 ;$$

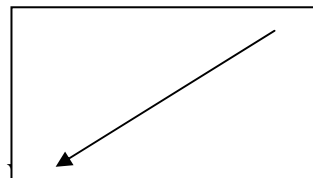


$$x = x + 1 ;$$

$$y = y + 1 ;$$



$$y = y + 1 ;$$

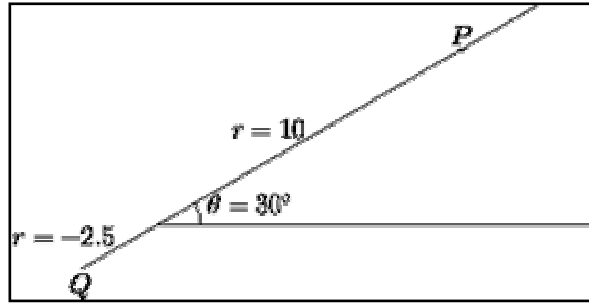


$$y = y - 1 ;$$

فكما تلاحظ أننا عند تغيير قيمة المتغير x أو المتغير y أو كلاهما يتغير موقع النقطة ولو لاحظت أن الرقم "1" الظاهر في كل المعادلات السابقة يعني أننا نقوم بتغيير الموقع نقطة واحدة ونستطيع طبعاً أن نغيرها إلى نقطتين أو ثلاث ... الخ. ولكن كل هذه المعادلات تعطينا القدرة على تغيير الموقع بشكل مستقيم فقط ونتساءل هنا ماذا نستطيع أن نفعل لو أردنا تغيير الموقع بشكل دائري مثلا كأن يقوم ممثل بالدوران نصف دائرة بدل الخط المستقيم ، في هذه الحالة لن تنفعنا المعادلات السابقة ولكن نستطيع استخدام النظام بولر لحل هذه المشكلة.

Polar Coordinates in the Plane (نظام بولر):

لنفرض أننا نريد وصف موقع النقطة P على سطح معين (الشاشة في حالتنا) كما عرفنا أننا في النظام السابق نستطيع وصفها باستخدام متغيرين x و y . وكذلك في نظام البولر نستطيع وصف النقطة P عن طريق استخدام متغيرين: متغير r هو المسافة بين النقطة P ونقطة الأصل ثم متغير الزاوية θ وهي الزاوية بين متجه المسافة الذي يصنعه المتغير r من النقطة P إلى نقطة الأصل وبين الإحداثي السيني (انظر إلى الشكل). (القيمة 360 أو 2π هي أعلى قيمة لزاوية θ وتعني دورة كاملة)

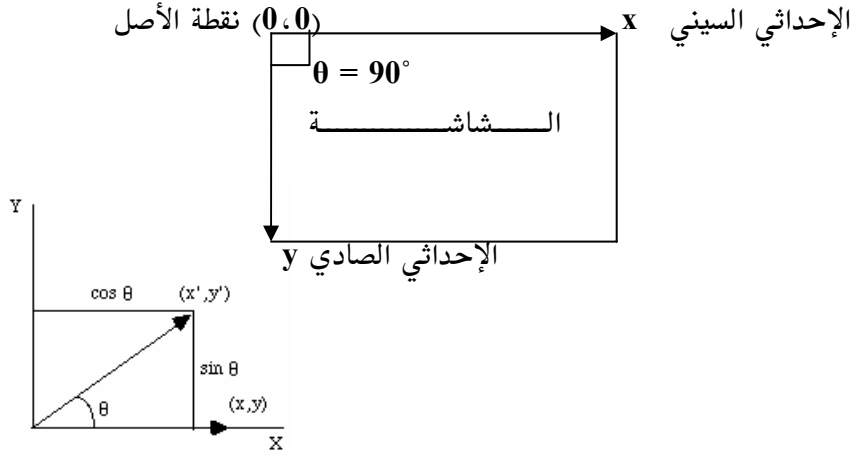


مثلا نستطيع أن نصف موقع النقطة P كما يلي : $(10, 30^\circ)$ أو $(10, 390^\circ)$ أو $(10, -330^\circ)$ لماذا ؟

لأن الزاوية 30° تساوي الزاوية $390^\circ = (360 + 30)$ وتساوي كذلك الزاوية $-330^\circ = (-360 + 30)$.

ما هي علاقة النظام بولر بالنظام السابقة (artesian) بمعنى أننا في برامجنا نحن نتعامل مع القيم x و y فقط وليس مع زوايا لذلك نحن نحتاج إلى معادلة نستطيع من خلالها تحويل المتغيرين r و θ إلى x و y وبالتالي استخدام هذا النظام الجديد في برامجنا.

لنفرض أن كلا النظامين السابقين يبدأان من نفس نقطة الأصل بالتالي نستطيع أن نقول أن الإحداثي الصادي في الجهة الموجبة يصنع زاوية قدرها 90 درجة مع الإحداثي السيني في الجهة الموجبة كذلك كما يلي :



وبالتالي نستطيع أن نقول أن النظامين لهما علاقة في وصف أي نقطة تحمل القيمة (r, θ) والقيمة (y, x) كما يلي :

$$\begin{aligned} x &= r \cos \theta, & r &= \sqrt{x^2 + y^2}, & \sin \theta &= \frac{y}{\sqrt{x^2 + y^2}}, \\ y &= r \sin \theta, & \theta &= \arctan \frac{y}{x}, & \cos \theta &= \frac{x}{\sqrt{x^2 + y^2}}. \end{aligned}$$

المختصر المفيد :

$$\begin{aligned} x &= r \cos \theta \\ y &= r \sin \theta \end{aligned}$$

الحركة المفصلة

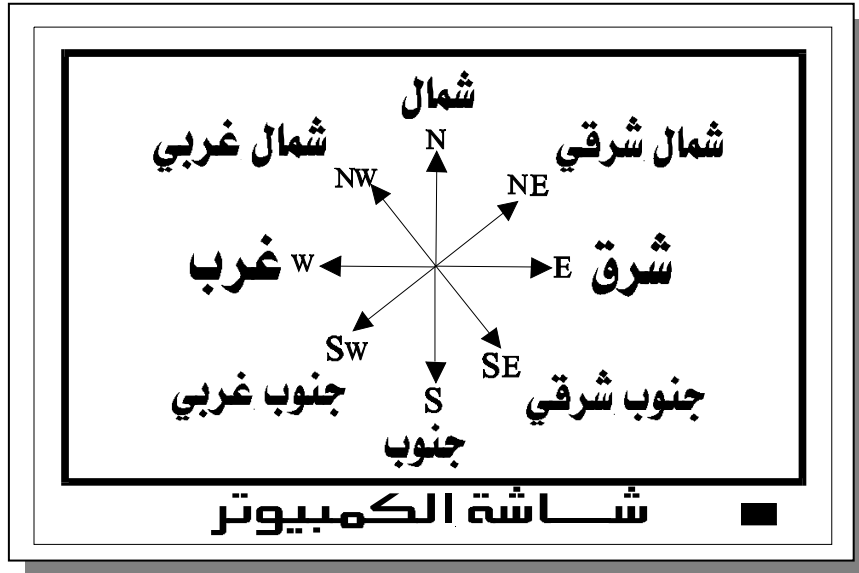
هذا كله مفيد لنا ، الآن نستطيع أن نتحكم في الحركة بشكل أكبر ، نستطيع أن نجعل أي نقطة تتحرك بشكل مستقيم أو دائري أو بزاوية معينة في أي جهة نشاء ، ولكننا أيضا مع هذا كله نلاحظ عدم قدرتنا على التحكم في الحركة بشكل أكثر تركيزاً وبشكل ابسط وكذلك بدون الدخول في معادلات رياضية كما فعالنا في الطريقتين الأوليتين ولهذا سوف ننظر إلى مصطلح وطريقة جديدة وفعالة سنطلق عليها "الحركة المفصلة" (هذا لا يعني أن الطريقتين السابقتين ليس لهما فائدة ، على العكس فعلى حسب حاجة البرنامج نستخدم الطريقة المناسبة. ونقوم هنا بالتعرف على افضل الطرق المستخدمة اليوم للحركة ، أما اختيار الطريقة المناسبة فهذا أمر يرجع للمبرمج).

الحركة المفصلة تعني أننا نستطيع أن نصف حركة نقطة معينة بشكل تفصيلي. لنفرض مثلا أن لدينا صحن فضائي نريده الدخول من منتصف يمين الشاشة بمعدل خمسين نقطة ثم يذهب إلى الأعلى بمعدل عشرين نقطة ثم إلى الأسفل ثم إلى الخ... الخ بحيث نصف الحركة كما نشاء وبشكل تفصيلي وحيث يستطيع البرنامج إعطاء تلك الأوامر الحركية لهذا الصحن الطائر لإتباعها بدون استخدام أي معادلات رياضية عندها نستخدم الحركة التفصيلية.

الحركة التفصيلية شديدة الفعالية بحيث نستطيع أن نجعل الحركة على كونها تفصيلية ولكن فيها نوع من الذكاء وبحيث تعتمد كذلك على الظروف المحيطة ، مثلا لعبة مصارعة الشوارع "Street Fighter 2" حيث يقوم اللاعب بحركة معينة والخصم يقوم بحركة أخرى تكون رد فعل لحركة اللاعب ، وهنا باستخدام الحركة التفصيلية نستطيع القيام بذلك وبشكل فعال.

لا بد انك عزيزي القارئ تسأل نفسك : ولكن كيف أستطيع أن أبرمج هذه الحركة ؟

بما أننا نتعامل مع الوظائف والكائنات في طريقة برمجتنا فإننا سوف نحتاج إلى كائن يحتوي على عدة وظائف تقوم بالتحكم في الحركة وكل ما علينا فعله هو إعطاء وصف للحركة المطلوبة ويقوم الكائن بدورة بالاهتمام بالباقي. قبل أن نبدأ في كتابة هذا الكائن للحركة التفصيلية سوف نقوم أول بتفصيل الشاشة وتقسيمها للجهات المطلوبة. وهي الشمال ، الجنوب ، الشرق و الغرب وذلك حتى نستطيع أن نصف اتجاه الحركة وكما هو واضح من الشكل في أسفل الصفحة فإن الشمال يعني إلى أعلى الشاشة والجنوب إلى أسفلها والشرق إلى يمين الشاشة والغرب إلى يسارها. بذلك نحن نستطيع وصف الحركة كما لو أننا نصف الطريق لشخص يبحث عن مكان معين.



مقدارها 5) كما يلي :

```
int pattern_2[] = {W, 25,
                  NW,5,N,5,NE,5,
                  N,5,NW,5,W,25,NW,5,
                  N,5,NW,5,W,25,
                  NW,5,N,5,NE,5,E,25,
                  NE,5,N,100};
```

أي أن الكرة تتجه لجهة الغرب (w) لمدة 25 (وحدة زمنية) ثم تتجه لجهة الشمال الغربي (NW) لمدة 5 ثم جهة الشمال (N) لمدة 5 ثم تتجه لجهة الشمال الشرقي (NE) لمدة 5 ثم جهة الشمال (N) لمدة 5 وهكذا بعد ذلك نضعها في المتغير pattern_2[] وهو مجرد متغير من نوع (int) نحن أطلقنا عليه هذا الاسم يحمل جميع الحركات التي قمنا بها للكرة وهكذا فصلنا الحركة فيه.

الآن لا بد انك عزيزي القارئ متشوق لترى كيف نستطيع أن نستخدم هذه المعلومات في برنامجنا. أولاً سنفرض بأن كائن الحركة التفصيلية (وأطلقنا عليه الاسم Pattern) موجود وكل ما نحتاج هو استخدامه فقط (إذا كنت مهتم في كيفية برمجته سوف ندخل في تفاصيل إنتاجه فيما بعد). في الوقت الحاضر سوف نفرض وجوده ونرى كيف يمكننا استخدامه مع مكتبه AGDX.

ملاحظة : المثال موجود من ضمن أمثلة القرص المدمج الملحق بالكتاب
(AGDX+patterns)

❖ أولاً نقوم في ملف التعاريف الرئيسي **main.h** بتعريف كائن الحركة كما يلي :

```
DECLARE Pattern Ball_Pat[30];
```

الآن أصبح لدينا كائن اسمه **Ball_Pat[30]** والرقم **30** يعني بأننا في الحقيقة أنشئنا ثلاثين كائن جميعهم يمثلون كائن الحركة المفصلة. والسبب في إنشائنا هذا العدد من الكائنات أننا في هذا المثال نستطيع أن نحرك ثلاثين كرة في نفس الوقت على الشاشة بدل كرة واحدة.

❖ ثانياً بعد أن أصبح لدينا مجموعة كائنات تمثل كائن الحركة. سوف نقوم في الملف الرئيسي **Main.cpp** بإنشاء كائن ممثل للكرة (هناك شرح خاص لكيفية إنشاء كائن الممثلين في مكتبته **AGDX** في فصل خاص به) ثم في الوظيفة الرئيسية **void Main(void)** نكتب ما يلي :

```
static time = 0;
if(--time <=0)
{
static xxx=0;
if(Input.Keys[DIK_UP] && xxx<=29)
{
Sprites.AddSprite(Ball, SPR_Ball,640,400,0,0,xxx++,0,0);
}
time = 20;
if(xxx ==30) xxx=0;
}
```

وكما نرى أن كل ما تقوم به هذه الشفرة ببساطة هو انه عندما نضغط على مفتاح السهم المتجه للأعلى من على لوحة المفاتيح فإن البرنامج يقوم بإنشاء وإظهار صورة للكرة وهذه الصورة لها رقم معين (من صفر إلى ثلاثين) نضعه في المتغير xxx فإذا وصل هذا الرقم إلى ثلاثين نقوم بإرجاعه مرة أخرى للصفر وهكذا.

❖ ثالثاً لننظر للوظيفة void SpriteBall(void) والمسؤولة عن إضافة ممثل الكرة كما يلي :

```
void SpriteBall(void)
{
    AGDXSprite* Node;
    AGDXSprite* Save;

    // Loop the list and update the sprites
    for(Node = Sprites.Next(); Node != Sprites.List(); Node = Save)
    {
        Save = Node->m_Next;

        switch(Node->m_Type)
        {
            case SPR_Ball:
            {
                Ball_Pat[Node->m_State].AI();
                Node->m_PosX += (Ball_Pat[Node->m_State].PostionX*4);
                Node->m_PosY += (Ball_Pat[Node->m_State].PostionY*4);
                Node->SetFrame(0);

                if(Node->m_PosY < -42 || Node->m_PosX < -42)
                {
                    Sprites.DelSprite(Node);
                    Ball_Pat[Node->m_State].Rest_AI() ;
                }
            }
        }
    }
}
```

(أرجو الملاحظة انك إذا لم تكن تفهم القسم المتعلق بكائن الممثل في المكتبة AGDX عليك مراجعته في الفصل الخاص به، هنا لن نتطرق لشرحه ولكن فقط لكيفية استخدام كائن الحركة التفصيلية)

كل ما سنقوم به في هذه الوظيفة هو التحكم في موقع الكرة السيني والصادي $(Node \rightarrow m_PosX)$ $(Node \rightarrow m_PosY)$ عن طريق إضافة المتغيرين الأعضاء لكائن الحركة التفصيلية وهما $(PostionX*4)$ و $(PostionY*4)$ وضربناهما بالرقم أربعة (وجدت هذا الرقم عن طريق التجربة والخطأ وتستطيع أن تغيره بنفسك لترى ما يحدث في المدة الزمنية للحركة).

دعونا نتحدث قليلاً عن المتغير $Node \rightarrow m_State$ وهو كما ترى متغير عضو في كائن الكرة (الممثل) وكلما أنشأنا كائن للكرة في الوظيفة الرئيسية $(void Main())$ فإننا نعطي كل كرة رقم خاص بها وبعد ذلك ننشئ كائن الحركة المفصلة لكل كرة حتى تكون حركتها مستقلة عن باقي الكرات.

وأخيراً في هذه الوظيفة نقوم بفحص موقع الكرة فإن كان أقل من (-42) في إحدى الإحداثيين السيني أو الصادي فإن الوظيفة تقوم بقتل كائن الكرة صاحبة ذلك الرقم عن طريق الأمر $(Sprites.DelSprite(Node))$ ثم لا ننسى أن نقوم بإرجاع القيم للمتغيرات الأعضاء لكائن الحركة لقيمتها البدائية عن طريق الأمر التالي :

$(Ball_Pat[Node \rightarrow m_State].Rest_AI())$.

ملاحظة : إذا لم تفهم أحد هذه الأوامر فأنصحك بتجربة المثال أولاً ثم قم بحذف ذلك الأمر وأعد إنتاجه وجربه لترى الفرق. وبذلك سوف تعرف عمل كل أمر.

❖ رابعا و أخيرا نقوم بإضافة المتغير []int pattern_2 ومحتوياته إلى الملف

هذا كل ما نحتاج معرفته لاستخدام كائن الحركة التفصيلية. أما بقية هذا الفصل فهو لشرح كيفية برمجة هذا الكائن وإن لم تكن تهتمك التفاصيل فيمكنك الاستفادة منه واستغلاله في برامجك بمعرفة شرحناه لهذه النقطة فقط. وتستطيع أن تضيف أو تغير الحركة في المتغير السابق كما في المثال الموجود على القرص الملحق لترى التغيير بنفسك. وإذا أردت الاستفادة واستخدام كائن الحركة فكل ما عليك هو الحاقه ببرنامجك.

من ماذا يتكون كائن الحركة class Pattern ؟

لنطلع أولا على الملف المحتوي على تعريف كائن الحركة Pattern.h والوظائف التابعة له والمطلوبة للقيام بالمهمة :

```
class Pattern
{
    public:
        Pattern();
        void Rest_AI();
        void AI();

        int PostionX;
        int PostionY;

        int ip , // pattern instruction pointer
        counter, // counter of pattern control
        pattern_index; // the current pattern being executed
        int *curr_pattern;
};
```

هذه هي محتويات ملف تعريف هذا الكائن وكما نرى انه يحتوي على ثلاث وظائف وعدة متغيرات سوف نلقي النظر عليها.

أولا لننظر إلى الوظائف الأعضاء لهذا الكائن :

1. وظيفة الباني Pattern() والتي سنقوم من خلالها على إعطاء القيم الأولية للمتغيرات المستخدمة في الكائن.

2. ثم الوظيفة Rest_AI والتي سنقوم من خلالها على إعادة كل المتغيرات لحالتها الابتدائية عند الانتهاء من الحركة المطلوبة.

3. وأخير الوظيفة AI والتي سنقوم بالعمل المطلوب لتحريك الموقع الحركي.

الآن لنرى الملف pattern.cpp والذي يحتوي على كائن الحركة التفصيلية :

```
#include "main.h"

//      N
//  NW ^ NE
//      \|/
//  W<---|--->E
//      /|\
//  SW v SE
//      S

enum
{
// pattern instruction codes for bot
E, // = 0  move west
NE, // = 1  move northeast
N, // = 2  move north
NW, // = 3  move northwest
W, // = 4  move west
SW, // = 5  move southwest
S, // = 6  move south
}
```

```

SE, // = 7  move southeast

// special instructions
STOP, // = 8  stop for a moment
RAND, // = 9  select a random direction
};

#define END -1 // end pattern
#define NUM_PATTERNS 2 // number of patterns in system

// patterns in code random format
int pattern_1[] = {W, 10, NW, 10, N, 10, NE, 10,
                  E, 10, SE, 10, S, 10, SW, 10,
                  W, 10, RAND, 10,

                  W, 20, NW, 10, N, 20, NE, 10,
                  E, 20, SE, 10, S, 20, SW, 10,
                  W, 10, END,0};

int pattern_2[] = {W, 25,
                  NW,5,N,5,NE,5,
                  N,5,NW,5,W,25,NW,5,
                  N,5,NW,5,W,25,
                  NW,5,N,5,NE,5,E,25,
                  NE,5,N,100};

// master pattern array
int *patterns[NUM_PATTERNS] = {pattern_1, pattern_2};

Pattern::Pattern()
{
    PostionX=0;
    PostionY=0;
    ip =0; // pattern instruction pointer for bot
    counter=0;
    curr_pattern= patterns[1]; // current pattern being processed
}

```



```

// As it say, Rest the AI
void Pattern::Rest_AI()
{
    PostionX=0;
    PostionY=0;
    ip =0; // pattern instruction pointer for bot
    counter=0;
    curr_pattern= patterns[1];
}

////////////////////////////////////

void Pattern::AI()
{
// this function controls the ai of the bot and the pattern
// processing

static int code, // current pattern code
        random; // current random

// test if it's time to process a new instruction
if (curr_pattern==NULL)
{
// select a random pattern in pattern bank
pattern_index = rand()%NUM_PATTERNS;
curr_pattern = patterns[pattern_index];

// now reset instuction pointer
ip = 0;

// reset counter
counter = 0;

} // end if

// process next instruction if it's time
if (--counter <= 0)
{
// get next instruction
code = curr_pattern[ip++];
random = curr_pattern[ip++];

// test what the code is
switch(code)
{

```

```
case E:
{
// set direction to east

                PostionX=1;
                PostionY=0;

// set counter to instuction random
counter = random ;
                Move_Time = random;

} break;

case NE:
{
// set direction to northeast

                PostionX=1;
                PostionY=-1;

// set counter to instuction random
counter = random ;

                Move_Time = random;

} break;

case N:
{
// set direction to north

                PostionX=0;
                PostionY=-1;

// set counter to instuction random
counter = random ;
                Move_Time = random;

} break;
```

```
case NW:
{
// set direction to northwest

PostionX=-1;
PostionY=-1;

// set counter to instuction random
counter = random ;
Move_Time = random;

} break;

case W:
{

PostionX=-1;
PostionY=0;

// set counter to instuction random
counter = random ;
Move_Time = random;

} break;

case SW:
{

PostionX=-1;
PostionY=1;

// set counter to instuction random
counter = random ;
Move_Time = random;

} break;

case S:
{
// set direction to south

PostionX=0;
PostionY=1;
```

```
// set counter to instuction random
counter = random ;
Move_Time = random;

} break;

case SE:
{
// set direction to southeast

                PostionX=1;
                PostionY=1;

// set counter to instuction random
counter = random ;
                Move_Time = random;

} break;

case STOP:
{
// stop motion

// set counter to instuction random

                PostionX=0;
                PostionY=0;

counter = random ;
                Move_Time = random;

} break;

case RAND:
{
// set direction randomly

                PostionX= 1+rand()%1 -(rand()%1);
                PostionY= 1+rand()%1 -(rand()%1);

// set counter to instuction random
counter = random ;
Move_Time = random;

} break;
```

```

case END:
{
// stop motion

// select a random pattern in pattern bank
pattern_index = rand()%NUM_PATTERNS;
curr_pattern = patterns[pattern_index];

// now reset instuction pointer
ip = 0;

// reset counter
counter = 0;

} break;

default: break;

} // end switch
}
}

```

محتويات الملف pattern.cpp

الآن لنبدأ بشرحه من البداية :

```

enum
{
E,
NE,
N,
NW,
W,
SW,
S,
SE,

// بعض الأوامر الخاصة
STOP,
RAND,
};

#define END -1
#define NUM_PATTERNS 2

```

أولاً قمنا بتقسيم الجهات وإعطاء كل منها رقم (هذا عمل الأمر enum في لغة السي المحسنة ، يقوم بإعطاء أرقام للكلمات حسب ترتيبها بداية من الرقم صفر). ثم استخدمنا الأمر **define** (وهذا أمر مشابه للأمر السابق) لإعطاء الاسم **END** القيمة 1- والاسم **NUM_PATTERNS** القيمة 2 (هذا الرقم يدل على عدد المجموعات الحركية المستخدمة).

بعد ذلك قمنا بكتابه الأوامر الحركية المطلوبة ووضعها في المتغيرات (استخدمنا متغيرين على سبيل المثال وسوف نقوم باستخدام أحدهما والذي سبق وشرحناه وهو المتغير `int pattern_2[]`).

```
int *patterns[NUM_PATTERNS] = {pattern_1, pattern_2};
```

بعد ذلك كما نرى في هذا الأمر أننا قمنا بإنشاء مؤشر للحركة المطلوبة وبما أننا استخدمنا مجموعتين من الحركة فإننا سنجعل المؤشر يشملهما.

```

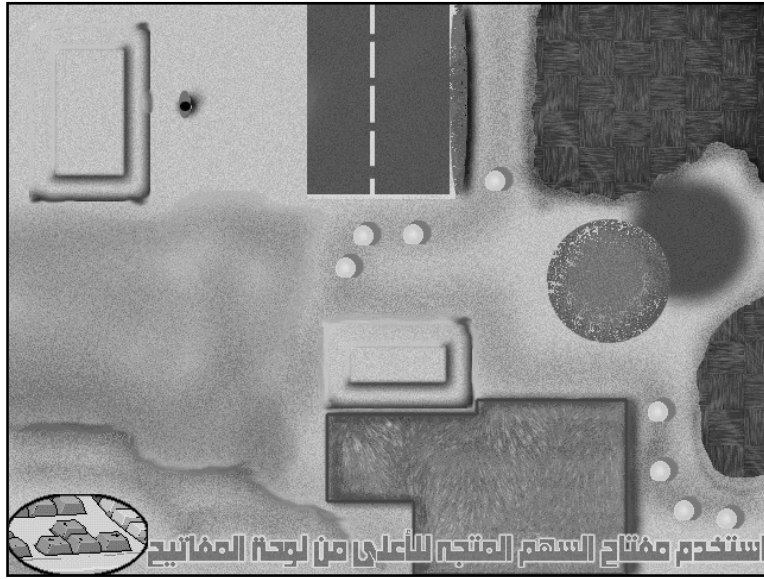
Pattern::Pattern()
{
    PostionX=0;
    PostionY=0;
    ip =0;
    counter=0;
    curr_pattern= patterns[1];
}

void Pattern::Rest_AI()
{
    PostionX=0;
    PostionY=0;
    ip =0;
    counter=0;
    curr_pattern= patterns[1];
}

```

أما في كل من الباني والوظيفة (Rest_AI) قمنا بإعطاء القيم الأولية للمتغيرات وقد تسأل نفسك هنا "إذا كنا قد فعلنا ذلك في الباني فلماذا نحتاج إلى الوظيفة (Rest_AI) لتقوم بنفس العمل مرة أخرى؟" لأن وظيفة الباني تُستدعى عند بداية استخدام الكائن ولكننا سوف نحتاج لوظيفة مشابهة وهي (Rest_AI) للقيام بنفس العمل في أي وقت وليس فقط عن بداية استخدام الكائن.

بالنسبة للوظيفة الرئيسية في هذا الكائن وهي الوظيفة (AI) فما تقوم به هذه الوظيفة هو التعرف على مجموعة الحركة الحالية (تذكر أننا قمنا بكتابه المجموعة الحركية للمتغير [pattern_2] والمتغير [pattern_1] في ما سبق في هذا الفصل) بعد ذلك تقوم الوظيفة وباستخدام الفعل الشرطي switch بالقيام بكل حركة في المجموعة. وقد تتساءل عزيزي القارئ لماذا يوجد أكثر من مجموعة حركية ولماذا تكلمنا فقط عن واحدة منها ([pattern_2])؟ والسبب أن هذا الكائن يعطينا القدرة على استخدام أكثر من مجموعة حركية. لنعود للمثال الذي ذكرناه سابقا وهو لعبة مصارعة الشوارع و أردت عزيزي المبرمج أن تقوم بكتابه مجموعة حركية لكل ردة فعل للمصارع. عندها سوف تكتشف انك تملك مجموعات كثيرة لكل حركة ، ولكنك تعرف أن كل مجموعة هي رد فعل لحركة معينة من الخصم. وبالتالي تستطيع أن تأمر هذا الكائن بأن يقوم باستخدام المجموعة المناسبة في الوقت المناسب.



الشاشة الرئيسية لمثال الحركة

Bryce 3D

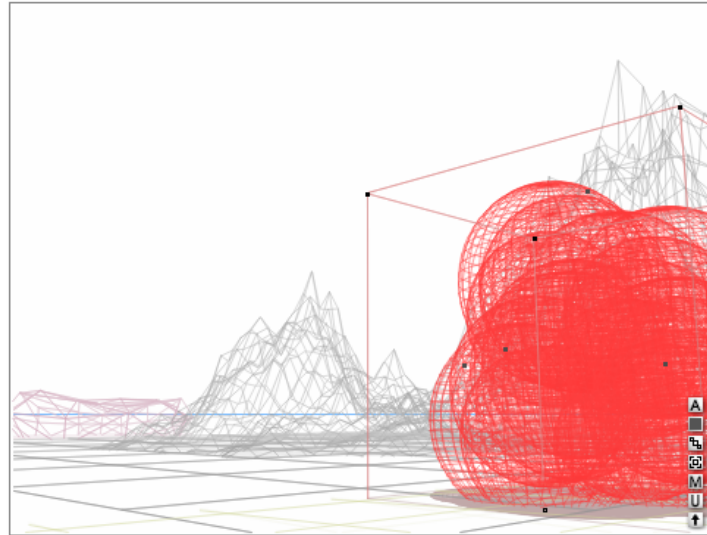
Bryce3D

هناك العديد من برامج الأبعاد الثلاثية في الأسواق ، وأغلبها يملك مواصفات متميزة لإنتاج الرسوم ذات الأبعاد الثلاثية ولكني لم أرى أية برنامج بهذا السعر (ما يقارب المائتين دولار فقط ، و تستطيع طلبه من أماكن مثل <http://www.buydirect.com> بحوالي \$150 وهذا السعر يعتبر معقول قياسا لهذه النوعية من البرامج) وهذه القدرة الهائلة في إنتاج المناظر الطبيعية ثلاثية الأبعاد.

برنامج **Bryce3D** (برايس ثري دي) من شركة **MetaCreation** (عنوان

الشركة على الإنترنت هو <http://www.metatools.com>) عندما أردت استخدام هذا البرنامج للمرة الأولى (توجد نسخة تجريبية على القرص المدمج الملحق بالكتاب تشمل جميع مميزات النسخة التجارية ماعدا بعض الأوامر كتخزين الملفات) توقعت بأن علي أن أقضي عدة ساعات في قراءة الكتب الملحقه له ثم توقعت بعد ذلك بأن أقضي ساعات أخرى إضافية لإنشاء مشهد ثلاثي بسيط (كما هي العادة مع البرامج ثلاثية الأبعاد) أبدأ من خلاله في تعلم إمكانيات البرنامج ولكن كانت دهشتي كبيرة لسهولة البرنامج ، لقد كان سهلا بحيث أنني استطعت إنشاء منظر طبيعي بمستوى فني رائع خلال دقائق. قد تتوقع عزيزي القارئ بأن ذلك كان ممكناً بسبب معرفتي السابقة ببعض ببرامج الأبعاد الثلاثية مثل 3D-Studio ولكني أحب أن أؤكد بأن هذا البرنامج مختلف تماما ولا يحتاج إلى أية معرفة مسبقة لبرامج الأبعاد الثلاثية لكي تستطيع إنتاج مناظر طبيعية خلال دقائق.

ولكن نظرة سريعة إلى طريقة عمل هذا البرنامج ستسهل على الكثيرين كيفية التعامل مع إمكانيات هذا البرنامج (لاحظ بأن واجهة البرنامج باللغة الإنجليزية ،



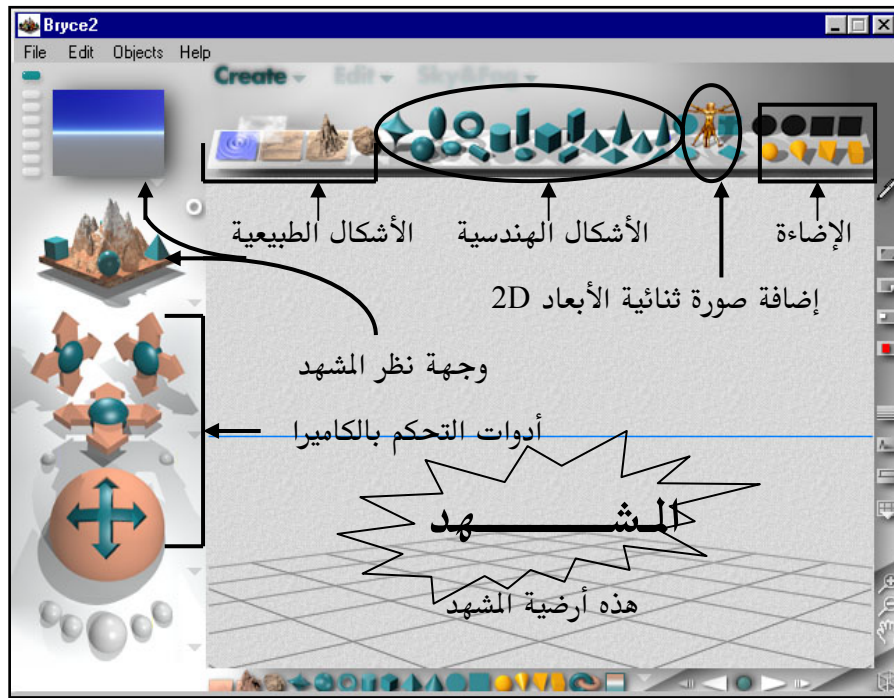
الشكل (1)



الشكل (2)

بعد قراءة هذا الفصل سوف تتعرف على الإمكانيات الأساسية للبرنامج ولن تكون اللغة مهمة بعد ذلك).

كما ونستطيع جلب واستخدام أنواع كثيرة من الأجسام ثلاثية الأبعاد إلى هذا البرنامج (يتضمن ذلك نوع 3ds لبرنامج 3d-studio و DXF لبرنامج AutoCAD). من النظر إلى الشكل (1) في الصفحة السابقة نستطيع أن نرى أنه بمجرد وضع الأجسام المطلوبة في المشهد (مثل الجبال والأشجار والمياه وغير ذلك من الأشكال العديدة المتوفرة لنا من خلال هذا البرنامج) نقوم باختيار الألوان المطلوبة (المقصود هو اختيار الشكل الخارجي) لكل جسم ويتم ذلك بسهولة وكما نرى فإن البرنامج يقوم بإظهار الشكل المطلوب. فمثلا النتيجة الأخيرة لشكل(1) نراها في الشكل (2).



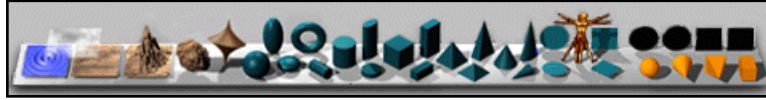
عند الانتهاء من تركيب برنامج Bryce3D "برايس ثري دي" من على القرص المدمج و إلى جهاز الكمبيوتر (هناك اختيار "إزالة تركيب أو uninstall" في حالة

لو أرت إزالة البرنامج من الجهاز) ثم تبدأ في تشغيل البرنامج عن طريق اختياره من قائمة البرامج وسوف يكون كما في الشكل السابق.

: Create

نرى في القائمة العلوية للشاشة الرئيسية للبرنامج ثلاثة أوامر مهمة وهي **Create** وتعني إنشاء و **Edit** وتعني تعديل ثم نرى **Sky&Fog** وتعني السماء و الضباب. وهذه أهم أوامر هذا البرنامج.

في البداية سوف نشرح الاختيار الأول وهو اختيار الإنشاء. عند الضغط على أحد هذه الأوامر فإن قائمة بأشكال معينة تظهر أسفله، مثلا في الاختيار الأول (اختيار الإنشاء) نرى القائمة التالية وهي مجموعة أشكال تمثل الأوامر المتوفرة تحت هذا الاختيار وهناك إمكانيات كثيرة لهذا الأمر :

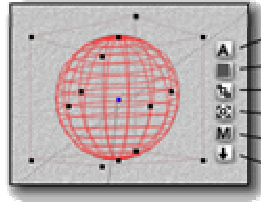


فعن طريق الضغط بإشارة الفأرة على الشكل المطلوب يقوم البرنامج بوضعه في المشهد الرئيسي وكما ترى من جهة اليسار إلى اليمين هناك مياه ، سحب، ارض، جبال و صخر ثم أشكال هندسية مختلفة مثل مربع و مستطيل ... الخ ثم بعد ذلك نرى رجل ذهبي اللون وهذا الشكل له استخدام خاص

سوف نتحدث عنة بعد قليل (يستخدم لإضافة الصور ثنائية الأبعاد 2-D) ثم أخيرا نرى أربع أشكال هندسية (في أقصى اليمين) وهذه الأشكال هي عبارة عن أنواع الإضاءة المطلوب استخدامها.

أما في جهة اليسار من الشاشة الرئيسية فإننا نرى مشهد جبلي ثم بعض الأسهم. ومن خلال هذه الأدوات تستطيع أن تتحكم في حركة الكاميرا وتستطيع أن تغير وجهة نظرك للمشهد بحيث تستطيع أن ترى المشهد من خلال الكاميرا أو من أعلى أو من الأسفل أو من جهة اليمين أو اليسار وهكذا. مثلا السهم المتجه إلى الأعلى يعطيك إمكانية تحريك الكاميرا إلى تلك الجهة ونفس الطريقة مع الأسهم الأخرى. وعن طريق الضغط على الجبل فإنك تستطيع أن تغير وجهة النظر إلى الجهة المطلوبة.

الآن كيف نقوم بإنشاء منظر خلفي حسب مشيئتنا؟



هذا أمر بسيط. نقوم باختيار الأشكال المطلوبة عن طريق الضغط عليها ثم ترتيبها حسب مشيئتنا. ولكن كيف نضع

الألوان المعينة للأجسام؟ مثلا كيف نقوم بإعطاء جبل أو كرة لون معين؟ بسيط أيضا، عند إنشاء الجسم المطلوب أو عند اختياره (عن طريق الضغط عليه بإشارة الفأرة) فإن لونه سيتغير إلى الأحمر الفاتح و تظهر لنا على اليمين قائمة الاختيار:

- **Attributes** تعني مميزات الجسم مثل حجمه وموقعه و علاقته بالأجسام الأخرى في المشهد وغير ذلك ، وفي أغلب الأحيان لا تحتاج أن تقلق لهذا الأمر.

- بعد ذلك ترى الأمر **Family** ويعني عائلة وكل ما يقوم به هو إعطاء الجسم المعين لون الإطار السلبي بحيث نستطيع أن نميز الأجسام في المشهد

إلى إطارات ذات لون معين فيسهل التعرف عليها في حالة وجود الكثير من الأشكال في مشهد واحد.

- بعد ذلك نرى الأمر **Link to object** (اربط بالجسم) وبكل سهولة يقوم هذا الأمر بربط أي جسم بجسم آخر في المشهد. ويتحرك معه إذا تحرك ويتأثر بجميع ما يتأثر به الجسم الآخر مثل تغيير في الحجم أو الموقع أو الجهة الخ.
- بعد ذلك نرى الأمر **Tracking** هذا الأمر شبيه جدا بالأمر السابق ويختلف عنه في أننا عندما نرتبط بجسم آخر عن طريق هذا الأمر فإننا لا نتحرك بتحريك الجسم الآخر ولكن نتجه تجاهه , مثلا نستطيع أن نستخدم هذا الأمر مع الكاميرا لربطها بجسم معين بحيث تتجه تجاهه ولكنها في نفس الوقت لا تتحرك من مكانها بمجرد تحرك الجسم الآخر.
- ثم بعد ذلك يأتي الأمر **Edit Materials** (تعديل الشكل الخارجي للجسم) فباستخدام هذا الأمر نستطيع أن نحدد نوع الجسم عن طريق تغيير الشكل الخارجي
- ثم أخيرا نرى **Land Object** (خفض الجسم) يقوم هذا الأمر بخفض

الجسم إلى مستوى أرضية المشهد)

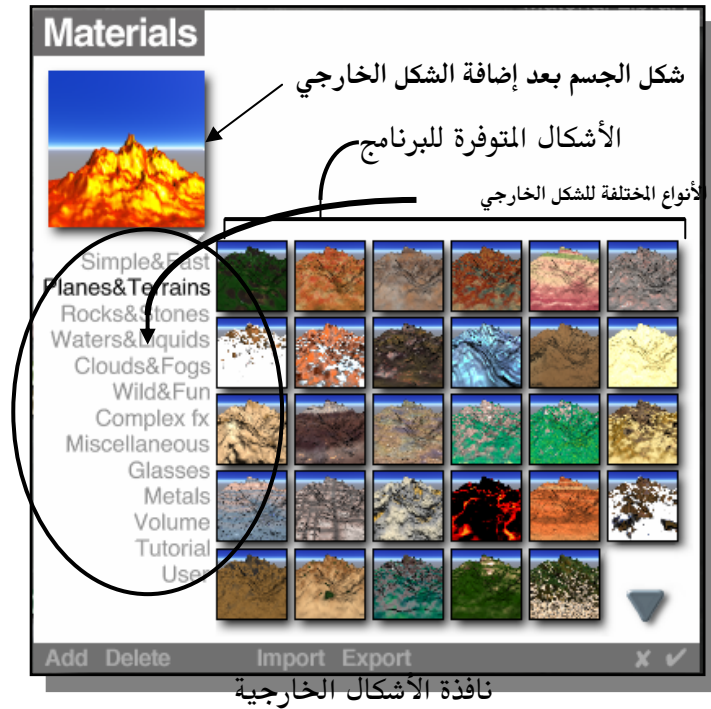


مختبر الأشكال الخارجية للأجسام

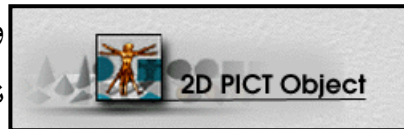
نقوم باختيار **Edit Materials** من القائمة حتى نستطيع تغيير الشكل الخارجي ، بعد الضغط على هذا الاختيار تظهر لنا شاشة جديدة (يطلق عليها مختبر الأشكال الخارجية للأجسام) تعطينا قدرة هائلة على التحكم بهذه العملية ناهيك عن سهولة الاستخدام.

لو نظرت إلى نافذة "مختبر الأشكال الخارجية للأجسام" في الشكل السابق فأنت في أغلب الأحيان لا تحتاج إلى أي تغيير وتقوم بقبول القيم المعطاة ، أحيانا يكون التغيير مهم (مثلا عند إنشاء الحركة تريد أن تقوم ببعض التعديلات على شكل بحيرة مثلا بحيث تتغير حركة الأمواج مع الوقت وهناك حالات أخرى كثيرة سوف تحتاج فيها لأجراء بعض التغييرات في مختبر الأشكال سوف تتعرف عليها مع الممارسة) فتضطر إلى القيام ببعض التعديلات. في أعلى اليسار من الشاشة السابقة سوف ترى صورة صغيرة تستخدم لإعطاءك الشكل الأخير للجسم بعد التعديل (في هذا المثال نرى صورة لجبل) بحيث تقوم بالنظر إليها لترى إن كان تعديلك هو المطلوب أم لا (توفر علينا هذه الطريقة الكثير من الوقت للتأكد من نوع الشكل المستخدم). هناك سهمان تابعان لهذه الصورة إحداهما متجه إلى الأسفل و الأخر إلى اليمين (راجع الشاشة السابقة لترى ذلك) ، تقوم بالضغط على السهم المتجه إلى اليمين فتظهر لك نافذة جديدة على الشاشة "نافذة الأشكال الخارجية".

عن طريق هذه النافذة الجديدة (انظر للشكل في الصفحة التالية) تقوم باختيار الشكل المطلوب للجسم وهناك الكثير من الأشكال الجاهزة للاختيار ولكن مع ذلك تستطيع أن تجلب "Import" أشكال أخرى كصور مثلاً في حالة احتياجك لها.



بعد أن تعرفنا على كيفية إنشاء المناظر الطبيعية والأشكال الهندسية دعونا ننظر إلى
الرجل الذهبي كثيرة نود لو أننا
ولماذا نحتاجه؟ أحيانا نستطيع إضافة شكل معين
إلى المشهد بدون الحاجة إلى إنشاء جسم ثلاثي الأبعاد هذا يوفر علينا الكثير من
الوقت وكذلك يعطينا القدرة على إدخال أي صورة ثنائية الأبعاد إلى المشهد (تصور
إضافة صورتك إلى المشهد مثلا).

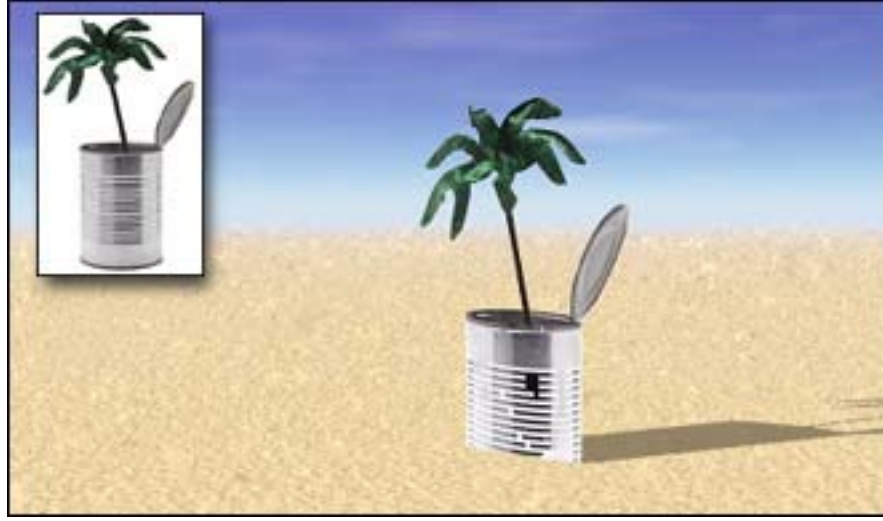


لكي نقوم بهذه العملية نقوم بالضغط على الرجل الذهبي بإشارة الفأرة بعد ذلك نرى النافذة التالية :



في هذه النافذة نقوم بإدخال صورة الشكل المطلوب بالإضافة إلى مسودة الصورة (كما هو الحال عند استخدام الممثلين في برمجة الألعاب فإن هناك لون محيط بالشكل المطلوب بحيث تقوم مسودة الصورة بإخبار البرنامج بإظهار المنطقة السوداء وإخفاء المنطقة البيضاء كما هو الحال في الشكل السابق فيقوم البرنامج بإعطاء الصورة الناتجة للشكل).

ويمكننا تخزين الصور في هذا المكان كذلك في حالة لو أردنا استخدام أكثر من صورة. دعونا نرى المثال التالي لتوضيح هذه العملية:



في هذا المثال استخدمنا صورة معينة (كما هي في أعلى اليسار) وبكل سهولة وضعناها في المشهد لأحظ أن المنطقة البيضاء أصبحت شفافة (لهذا السبب نحتاج إلى إنشاء مسودة لكل صورة نستخدمها) والشكل ككل أصبح مقنع للغاية حتى أنك تحسب أن الصورة هي في الأصل جزء من المشهد.



وأخيرا في قائمة "Edit" تأتي الإضاءة ،

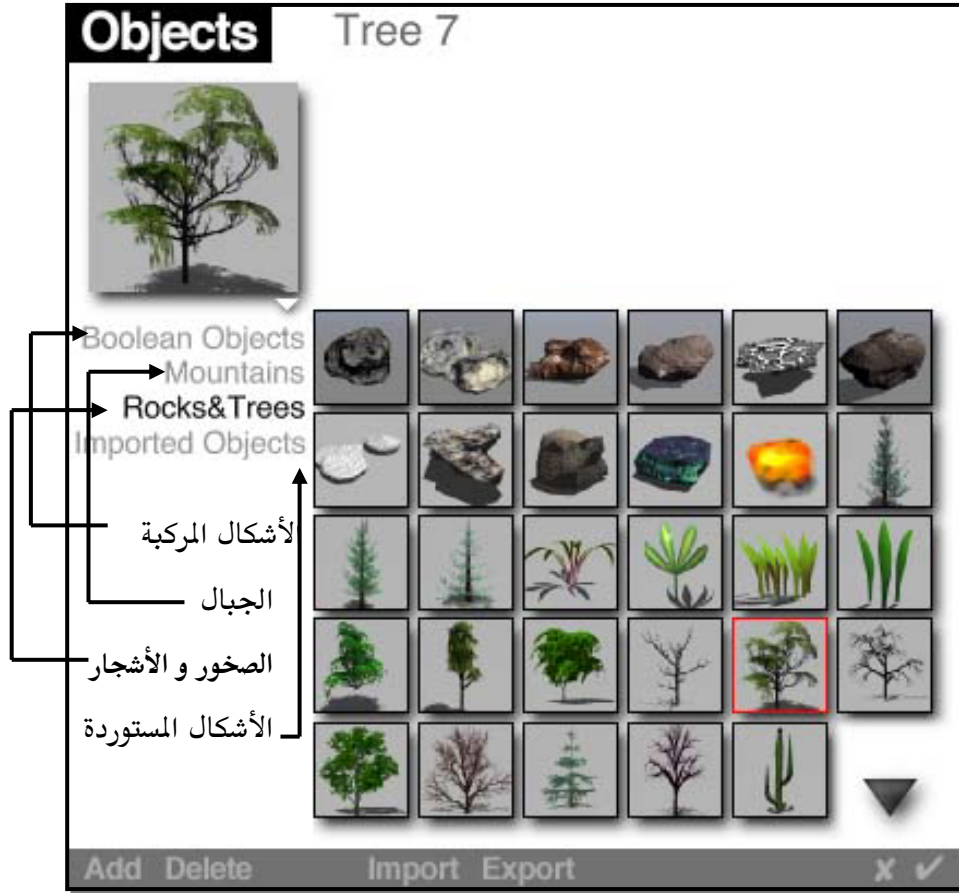
(انظر الأنواع المختلفة للإضاءة)

في هذه البرنامج تكون الإضاءة صادرة أما من الشمس أو القمر ولكن يمكنك إضافة أنواع مختلفة من الإضاءة لإعطاء تأثير معين في المشهد. مثلا إضافة كشافات لسيارة في المشهد أو إعطاء ضوء بعيد ساطع من مدينة في الأفق. وكما نرى من الشكل السابق وجود أربع أنواع من الإضاءة. (سوف نشرحها من اليسار اليمين حسب

ترتبتها) النوع الأول هو الإضاءة الشمسية (تتمثل في الشكل الكروي) وهذا النوع من الإضاءة يطلق الأشعة الضوئية في كل الاتجاهات، مثل الشمس الصغيرة، ونستطيع استخدام هذا النوع مثلا لإعطاء ضوء لمصباح معلق على حائط أو شمعة مشتعلة.... الخ. النوع الثاني والثالث هي الإضاءة المسلطة وتنتقل الأشعة الضوئية بها من المصدر إلى نقطة معينة في المشهد ومن الأمثلة على ذلك المصباح والفرق بين هذين النوعين هو فقط في شكل الإضاءة المسلطة، فالأول دائري والثاني مربع. أما النوع الأخير فهو الإضاءة المتوازية ويختلف عن النوعين السابقين في أن الأشعة الضوئية تكون بحجم المصدر مهما ابتعدت المسافة (كما هو معروف أن الإضاءة تبدأ من المصدر ثم تكبر في الحجم وتضعف كلما ابتعدنا عن المصدر) مثال على هذا النوع أشعة الليزر.

لقد تعرفنا في الأمر السابق "Create" على كيفية إنشاء مناظر طبيعية عن طريق إضافة الأشكال المطلوبة في المشهد. سواء أكانت تلك الأشكال عبارة عن جبال أو مياه أو سحاب أو أرض أو صخور بالإضافة إلى عدة أشكال هندسية يمكننا أن نستخدمها في المشهد وأيضا يمكننا استخدام صور ثنائية الأبعاد، كما سبق وشرحنا، وإضافة بعض المؤثرات إلى المشهد وكذلك إلى توفير الوقت في إضافة أشكال معينة حسب الحاجة. باستخدام هذه المميزات فقط نستطيع أن ننشئ الكثير من المناظر الطبيعية التي قد نتصورها. ولكن تظل كل هذه الإمكانيات محدودة لأنه توجد في الواقع أشكال لا نهائية نراها في الطبيعة (خذ مثلا الأشجار أو المباني) لا تحدد بالأشكال المتوفرة عن طريق هذه الأوامر لذلك وضعت هذه النقطة في الاعتبار عند تصميم البرنامج، فقد أعطانا هذه البرنامج إمكانيات هائلة في دمج الأشكال بعضها ببعض لإنتاج أشكال جديدة كما أنه وفر لنا إمكانية

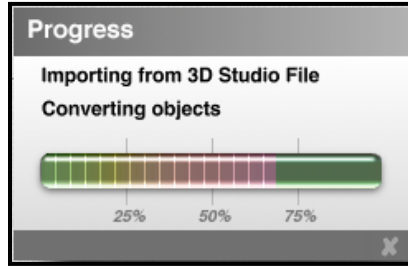
إنشاءها حسب مشيئتنا وكذلك إمكانية جلبها من برامج أخرى لاستخدامها في هذه البرنامج، ونستطيع أن نقوم بذلك عن طريق الضغط على السهم المتجه إلى الأسفل في الأمر Cerate عندها نستطيع أن نرى نافذة "Objects" التالية:



من خلال هذه النافذ نرى العديد من الأشكال المختلفة (ثلاثية الأبعاد) والتي تأتي بالمجان مع هذا البرنامج. نستطيع تقسيمها حسب مشيئتنا، مثلا سوف نرى عدة أنواع متوفرة (الأشكال المركبة ، الجبال ، الصخور و الأشجار ، الأشكال المستوردة) ونستطيع إضافة أشكال ثلاثية الأبعاد أخرى عن طريق الأمر "Add" و

إزالتها عن طريق الأمر "Delete". مثلا افرض أننا أردنا أن نضيف جسم ثلاثي الأبعاد تم إنتاجه في برنامج آخر ثم تم حفظه في ملف من نوع 3ds (نستطيع جلب الأنواع 3ds ، dxf ، t3d ، 3mf ، b3d و obj) فإننا نختار " Import " من قائمة "Object..." من أعلى الشاشة ثم نختار الملف المطلوب فتظهر لنا

الشاشة التالية :



ويقوم برنامجنا بتحويل الملف المخزن لنوعية الملفات المستخدمة حتى يتسنى لنا استخدامها في المشهد الحالي.

بعد ذلك عن طريق نافذة "Objects" السابقة نستخدم الأمر "Add" فيقوم بإضافة ذلك الجسم الجديد.

: Edit

الآن نعود إلى القائمة العلوية مرة أخرى (عند بداية الشرح تكلمنا عن الشاشة الرئيسية ثم تحدثنا عن الاختيار الأول "Create") وننظر إلى الأمر الثاني "Edit" أو تعديل ، بعد الضغط عليه بإشارة الفأرة تظهر لنا القائمة التالية :





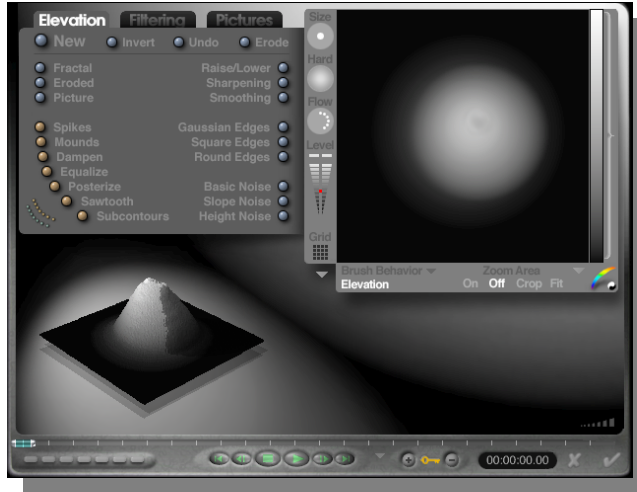
وكما تعودنا فأن هذا البرنامج يستخدم أشكال معينة كتعبير عن الأوامر المستخدمة به. مثلا في هذا القائمة نرى حسب الترتيب من اليسار إلى اليمين ما

يلي:

- هذا الشكل هو المسؤول عن الشكل الخارجي كما سبق وشرحنا ، فنستطيع عن طريق الضغط على هذا الشكل الذهاب إلى مختبر الأشكال الخارجية للأجسام (انظر إلى قائمة أوامر **Create** لتتعرف على مختبر الأشكال الخارجية).
- بعد ذلك نرى الشكل  وهذا الشكل هو المسؤول عن التحكم في حجم الجسم "Resizing" من جميع الاتجاهات (بما أننا نتعامل مع محيط ثلاثي الأبعاد فإن الاتجاهات تكون x و y و z).
- ثم نرى الشكل  ويمكننا من تدوير الجسم في كل الاتجاهات ونحتاج في كثير من الأحيان تدوير الجسم إلى اتجاه معين ، مثلا عند إضافة جسم ثنائي الأبعاد كم سبق وشرحنا (عن طريق الضغط على الرجل الذهبي تحت قائمة **Create**) فإننا نحتاج إلى تدوير الجسم ليقابل الكاميرا فنرى الشكل بشكل مقنع.
- بعد ذلك نرى الشكل  وهو الشكل المسؤول عن تحريك الأجسام في المشهد. بكل بساطة نختار الشكل المطلوب ونقوم بالضغط على هذا الأمر فيتحرك الجسم في الاتجاه المطلوب (هناك ثلاث اتجاهات كما سبق وشرحنا وهي x و y و z).
- بعد ذلك نرى الشكل  ونستخدم هذا الأمر في موازنة الأجسام لبعضها البعض ويجب عند استخدام هذا الأمر أن تكون اخترت شكلين بحيث تتم عملية الموازنة بينهما (لاختيار شكلين تقوم بالضغط على الشكل الأول بإشارة الفأرة ، بعد اختياره سوف يتغير لونه إلى الأحمر كما هو حال الأجسام

المختارة، ثم أضغط على الشكل الثاني بإشارة الفأرة بنفس الطريقة ولكن مع مفتاح Shift من لوحة المفاتيح).

- أما الأمر الذي يلي ذلك فهو غريب نوعاً ما. ويتمثل في الشكل  ويقوم بإلقاء الأجسام المختارة بشكل عشوائي في المشهد مع تغيير (أيضاً بشكل عشوائي) في حجمها و تدويرها. قد تسأل في ماذا سوف يُستخدم هذا الأمر؟ هناك أوقات يكون هذا الأمر فيها مفيداً. تخيل مثلاً أن هناك انفجار بركاني والأجسام تتقاذف من البركان إلى سطح الأرض بشكل عشوائي (سوف نتحدث عن إنشاء الحركة فيما بعد) بحيث تتغير أحجامها عند وصولها إلى السطح.
- وأخيراً نرى الأمر  ونستطيع من خلاله أن نتحكم في الشكل (ليس كل الأشكال) بحيث نغير فيه كما نشاء (يمكن كذلك الوصول إلى هذا الاختيار عن طريق اختيار E من قائمة الاختيارات (وهي القائمة التي تظهر على يمين الشكل عند اختياره وقد سبق وشرحنا أغلب أوامرها في البداية) عند الضغط عليه بإشارة الفأرة فإننا نرى نافذة جديدة:

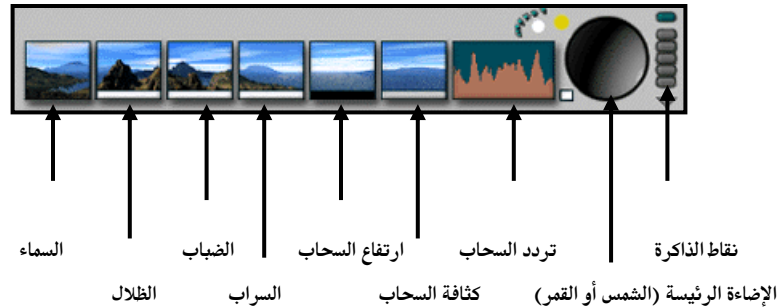


عن طريق هذه النافذة نتحكم في الصورة التي تظهر في أعلى اليمين. فمن خلال تغيير تلك الصورة تتغير النتيجة الأخيرة للشكل المطلوب وتكون نتيجة ذلك التغيير في أسفل اليسار. وكما نرى في هذا المثال كلما زادت المنطقة (في الصورة أعلى اليسار) اقتربها من البياض فإن الجسم (في أسفل اليمين) يكون عالي المستوى والعكس. من خلال هذه الطريقة نستطيع أن نكون أشكالاً كثيرة كما نريد، واحب أن انبه انك تستطيع كذلك رسم هذه النوعية من الصور في برنامج رسم وتجلبها إلى هذه الشاشة لاستخدامها. كذلك يأتي على القرص المدمج مع هذا الكتاب برنامج صغير يقوم بتحويل ملفات برنامج **Vistapro** (هذا برنامج آخر شهير مهتم بالمناظر الطبيعية وتأتي النسخة الأولى بالمجان مع كتابي السابق "برمجة ألعاب الكمبيوتر على النظام windows95") إلى صور من هذه النوعية بحيث نستطيع أن نستخدمها مع هذا البرنامج (الإصدار الرابع من **Bryce** يستطيع جلب ملفات برنامج **Vistapro** مباشرة بدون الحاجة لأي برنامج خاص).


Sky&Fog (السماء والضباب):


الآن نعود إلى القائمة العلوية مرة أخيرة وننظر إلى الأمر الثالث "Sky&Fog"


أو السماء والضباب، بعد الضغط عليهما بإشارة الفأرة تظهر لنا القائمة التالية:





من خلال هذا الأمر نستطيع أن نتحكم في عدة مؤثرات طبيعية للمشهد مثل السحاب، الضباب، السراب، كثافة السحاب، اتجاه الإضاءة (سواء من الشمس أو القمر) وكذلك شدتها و لونها.


نبدأ بالأمر الأول من اليمين وهو أمر السماء  ونستخدم هذا الأمر في التحكم في رسم السماء من حيث شدتها (إذا كانت مظلمة أم فاتحة اللون) وكذلك نستطيع أن نجعل صورة السماء غير متأثرة بالإضاءة الرئيسية ، أو أن نجعل السماء صافية (بمعنى أنها تحمل لون واحد معين كخلفية للمشهد).

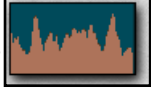
بعد ذلك نرى أمر الظلال  وهو الأمر المسؤول عن درجة التظليل في المشهد فعن طريق الضغط على هذا الشكل بإشارة الفأرة ثم تحريكها لجهة اليمين أو اليسار فإنك تتحكم في شدة التظليل (الفتاح أو الغامق) إذا كان هذا الأمر يساوي صفراً فإن جميع الأجسام المستخدمة في المشهد لن يكون لها ظلال (لاحظ أن تغيير اللون في هذا الأمر عن طريق الضغط على المستطيل الأبيض أسفل الشكل لا يغير لون الظلال ولكن لون الإضاءة الرئيسية في المشهد).


الأمر الثالث من أوامر Skey&Fog هو أمر الضباب ويتمثل في الشكل  وكما هو الاسم فإن هذا الأمر هو المسؤول عن درجة الضباب في المشهد كذلك عن ارتفاع الضباب عن سطح الأرض. عن طريق الضغط على هذا الشكل بإشارة الفأرة و تحريكها لجهة اليمين أو اليسار فإنك تتحكم في درجة الضباب أما عند تحريك الإشارة إلى الأعلى أو الأسفل فإنك تتحكم في ارتفاع الضباب.


الأمر الرابع هو أمر السراب (Haze) ويتمثل في الشكل  وعن طريق هذا الأمر نستطيع أن نتحكم في السراب بحيث نستطيع أن نجعل المشهد يختفي بالتدرج في الأفق حتى تلتقي الأرض بالسماء. بدون تغيير هذا الأمر يكون المشهد بلا أي سراب ويبدأ السراب في الظهور عن بعد. وهنا يمكننا التحكم في تلك الدرجة.


أما الأمر الخامس فهو أمر ارتفاع السحاب ويتمثل في الشكل  بحيث نستطيع أن نتحكم في ارتفاع السحاب عن سطح الأرض وكذلك في حجم السحاب وكما كان الحال مع أمر الضباب فإننا هنا عن طريق الضغط على هذا الشكل بإشارة الفأرة و تحريك لجهة اليمين أو اليسار فإنك تتحكم في حجم السحاب أما عند تحرك الإشارة إلى الأعلى أو الأسفل فإنك تتحكم في الارتفاع.

الأمر السادس من أوامر Skey&Fog هو أمر كثافة السحاب ويختلف عن الأمر السابق في انه بالإمكان التحكم في كثافة السحاب ويتمثل في الشكل  حيث نستطيع مثلاً أن نغطي السماء بطبقة كثيفة من السحاب أو نستطيع أن نجعل طبقة خفيفة من السحاب.

الأمر السابع هو التردد  نستطيع عن طريقة التحكم في عدد مرات تكرار نفس الشكل للسحاب في المشهد تحريك هذا الأمر عن طريق إشارة الفارة إلى اليمين مثلاً يقلل من حجم السحاب ولكن يزيد من عدد مرات تكراره والعكس صحيح. أما تحريك هذا الأمر إلى الأعلى أو الأسفل فإنك تستطيع أن تتحكم في شكل أطراف السحاب إذا كانت حادة بالنسبة للون المشهد أو العكس (لفهم هذا الأمر بشكل أفضل أو أي أمر نتحدث عنه هو تجربته مباشرة على جهاز الكمبيوتر لترى النتيجة في الحال).

أمر التحكم في الإضاءة الرئيسية وموقعها يتمثل في الشكل  عن طريق هذا الأمر نستطيع التحكم في الجهة التي يصدر منها ضوء الشمس أو ضوء القمر (على حسب اختيارك لنوعية الإضاءة الرئيسية) عن طريق تحريك هذا الشكل 360 درجة نستطيع أن نجعل الضوء يصدر في كل الاتجاهات في المشهد كما يمكننا التحكم في لون الإضاءة عن طريق اختيار اللون المرغوب اسفل هذا الشكل.

وأخيرا نرى أمر التحكم بنقاط الذاكرة عن طريق الأمر  عن طريق هذا الأمر نستطيع أن نحفظ جميع التغييرات التي قمنا بها في نقطة معينة بحيث نستطيع إجراء تغييرات مختلفة وحفظها على نقطة أخرى وبهذه الطريقة نستطيع أن ننتقل من تغيير إلى تغيير بطريقة أسهل. النقطة الأولى وضعت للبرنامج ولا يمكننا أن نستخدمها لحفظ تغييراتنا (لأنها دائما تحتفظ بالتغيرات الأساسية للبرنامج بحيث نستطيع الرجوع إليها في حالة احتياجنا لذلك)

⊙خدعة : نستطيع وضع الأشكال الخارجة على أي جسم مهما كانت ، مثلا لو أردنا إنشاء بحيرة فإننا نستطيع أن نستخدم جسم الجبل مع الشكل الخارجي للماء ثم نقوم بتعديل حجم الجبل (عن طريق اختيار الجسم ثم الضغط على الشكل  من أوامر Edit) لكي يأخذ شكل البحيرة.

من الإضافات المهمة لهذا البرنامج الحركة ، فنستطيع أن نقوم بإعطاء الحركة لأي جسم في البرنامج سواء كنا نود أن نصور بحيرة أو نار أو كنا نود إظهار الشمس في أوقات مختلفة من النهار). و نتحكم في الحركة عن طريق الأوامر في أسفل الشاشة الرئيسية للبرنامج:



الدخول في كيفية إنشاء الحركة في هذا البرنامج سوف يفتح علينا موضوعا آخر خارج مجال هذا الكتاب. لذلك سوف أترك لك عزيزي القارئ هذا الجزء لتكتشفه بنفسك.

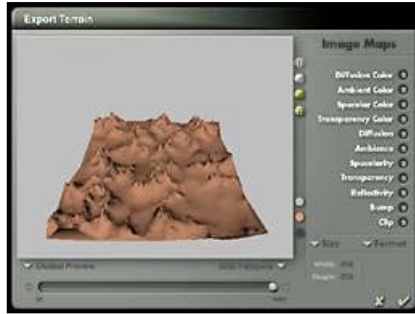
برنامج Bryce3D في رأيي هو أحد البرامج المهمة لكل مبرمج للصوت والصورة على الكمبيوتر وبرنامج مهم جدا لتصميم ألعاب الكمبيوتر بشكل خاص. ومع أننا حاولنا أن نتكلم عن أغلب مميزات وأوامر هذا البرنامج إلا أن الكثير بقي لتكتشفه بنفسك ولا أعتقد أن اللغة أصبحت مشكلة بعدما صارت أغلب أوامر واجهة البرنامج الرئيسية واضحة. (سوف استخدم هذا البرنامج في بعض الأمثلة في هذا الكتاب)

ملاحظة: لقد كتبت هذا الفصل بعد أسبوع من استعماله لهذا البرنامج وهذا دليل كافي لك ،عزيزي القارئ بأن البرنامج سهل الاستخدام وإن أستغرق بعض الوقت لتتعود عليه في بداية الأمر.

بعض المميزات الجديدة في الإصدار الرابع لـ Bryce3D

عندما كتبت هذا الفصل كان الإصدار الرابع قيد التطوير ، أما الآن وبعد أن أصبح الإصدار الرابع في الأسواق. وبعد تجربته تبين لي أن واجهة المستخدم والتي تحدثنا عنها لم تتغير ولكن تم إضافة بعض المميزات التي أود أن أذكرها ، من جهة أخرى لا زلت أرى أن الإصدار الثالث هو كل ما سوف تحتاجه. وبما أن الإصدار الرابع في الأسواق فهذا يعني أن الإصدار الثالث أصبح رخيص الثمن (اعتقد أنك تستطيع الحصول عليه بأقل من مائة دولار أمريكي).

المميزات الجديدة في الإصدار الرابع :



- تستطيع الآن حفظ الأجسام ثلاثية الأبعاد (جبال , سحب , انهار .. الخ) والتي ينتجها البرنامج على ملف لاستخدامها مع برامج الأبعاد الثلاثية الأخرى.

- واجهه جديدة للحركة , تسهل علينا استخدام وإنشاء الحركة ثم حفظها على ملف لاستخدامها حسب مشيئتنا بعد ذلك . ونستطيع حفظها باستخدام النوعيات

التالية :



1 . QuickTime (.MOV)

2 . QuickTime VR Panorama (.MOV)

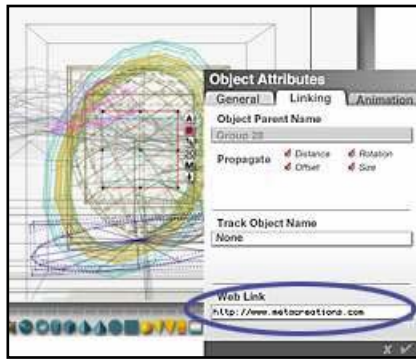
3 . RealMovie (.RM)

بالإضافة طبعا لجميع النوعيات التي يحفظها الإصدار الثالث.

- إضافات لعدد كبير من البرامج الثلاثية الأخرى ولكي يستطيع البرنامج تبادل الأجسام الثلاثية معها.



- إضافة إمكانية جديدة يطلق عليها مختبر السماء (Sky Lab) وهي نافذة عن طريقها نستطيع أن نتحكم في كل ماله علاقة بلون السماء في المشهد ومشاهدة التغيير مباشرة في هذه النافذة. كما نستطيع أن نتحكم في التغيير في السماء مع الوقت في حالة لو اردنا استخدام الحركة في المشهد.



- من الإضافات الجديدة كذلك في هذا الإصدار هو وضع الإنترنت في الحسبان ، الآن تستطيع أن تضع خريطة للصورة التي تنتجها على هذا البرنامج والتي تحتوي على وصلات لصفحات أخرى بدون الحاجة لأي برنامج إضافي للقيام بذلك. هذا بالإضافة إلى بعض المميزات الأخرى مثل استخدام الأفلام المتحركة التي تنتجها مباشرة لصفحات الإنترنت حتى يتمكن الآخريين من مشاهدتها. والتخاطب مع مستعملين آخرين لهذا البرنامج من خلال قناة خاصة يطلق عليها Bryce Talk. ثم القدرة على حفظ الأجسام الثلاثية الأبعاد باستخدام نوعية الملفات الجديدة "MetaStream" من نفس الشركة المصنعة والتي من خلالها يستطيع متصفح الإنترنت الإطلاع على الجسم الثلاثي وتحريكه بشكل مباشر.

الفصل التاسع

الحركة

وبرنامج Poser

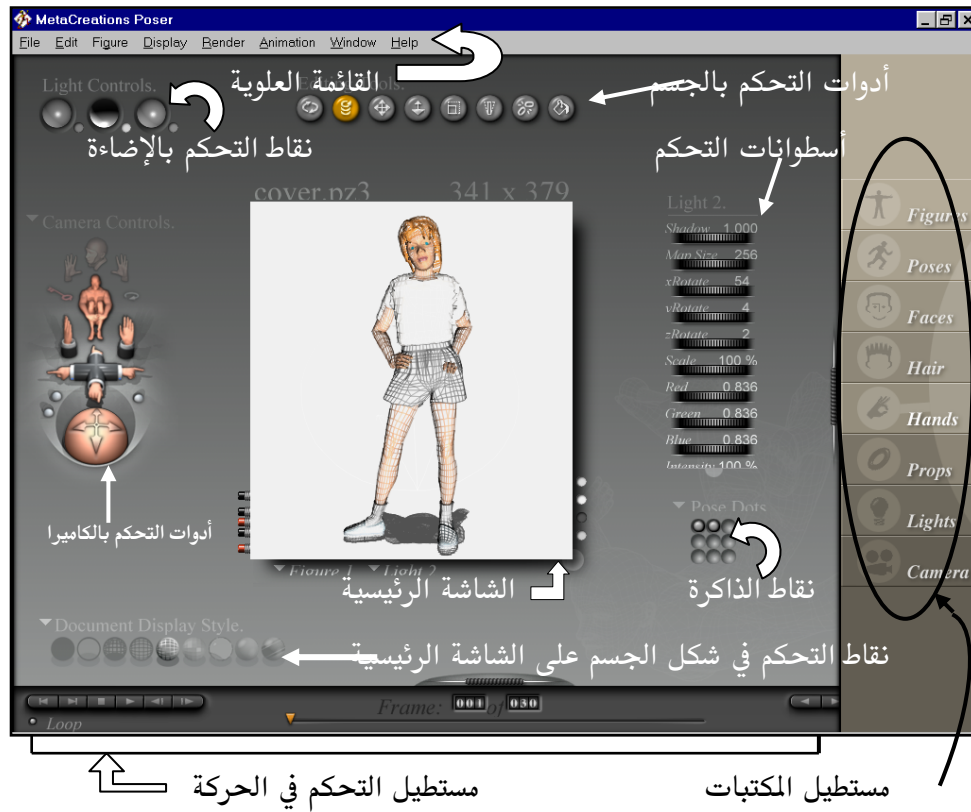
الحركة وبرنامج Poser

لم تجد شركة "MeatCreation" صعوبة لتثبيت وجودها كأحد انجح شركات صناعة برامج الأبعاد الثلاثية ، ولم لا وهي الشركة التي جعلت من التكنولوجيا الغالية الثمن والمكلفة آلاف الدولارات في متناول من يستطيع دفع اقل من مائتي دولار أمريكي فقط.

برامج إنشاء الحركة والمناظر الطبيعية ثلاثية الأبعاد على الكمبيوتر هي من البرامج الغالية الثمن ولا اخفي إعجابي الشديد بهذه الشركة التي أثبتت أنها تستطيع أن تجعل من البرامج التي تنتجها بثمان رخيص تتعادل بل وتتفوق أحيانا على شبيهاتها من البرامج ثلاثية الأبعاد والتي تكلف آلاف الدولارات. برنامج Poser3 (يطلق عليه "بوزر ثري") هو البرنامج الثاني في هذا الكتاب من الشركة نفسها التي أنتجت البرنامج السابق "برايس ثري دي" والذي سبق واطلعنا عليه. في البرنامج السابق قمنا بإنشاء المناظر الطبيعية بكل بساطة وخلال دقائق ، وقد تميز البرنامج السابق "برايس ثري دي" بسهولة الاستعمال ودقة التحكم. ونرى نفس المستوى في هذا البرنامج الجديد من نفس الشركة (Poser 3 هو الإصدار الثالث والرابع والمتميز كثيرا عن الإصدارات السابقة) سوف نرى سهولة إنشاء الأجسام البشرية ثلاثية الأبعاد (أطفال ، رجال ، نساء ، شيوخ... الخ وكذلك بعض الحيوانات كالخيل والكلب والديناصور وسمك الدولفين) ثم إعطاءها الحركة بشكل مبهر حتى انك تحسبها حية. كل هذا طبعا مهم لنا نحن كمبرمجين لأننا نحتاج إلى إنشاء الحركة في برامجنا. وقد كان إنشاء الحركة ثلاثية الأبعاد يتطلب برامج غالية الثمن مقارنة ببرامج هذه الشركة (مثل SoftImage من شركة Microsoft

والبرنامج الشهير 3D-Studio من شركة Autodesk) وحتى هذه البرامج كانت تتطلب مهارة في استخدامها للوصول للنتيجة نفسها والتي نستطيع إنشاؤها خلال دقائق في هذا البرنامج.

ونستطيع كذلك استخدام هذا البرنامج تقريبا مع جميع برامج الأبعاد الثلاثية وذلك عن طريق تخزين الملفات على النوعية المطلوبة (نستطيع كذلك أن نستخدم الأجسام ثلاثية الأبعاد مع العنصر Direct3D عن طريق تخزين الجسم المطلوب على ملف من نوعية 3DS ثم تحويله إلى ملف من نوعية X)



هذه هي الواجهة الرئيسية للبرنامج ونرى من خلالها كل الأدوات اللازمة لجعل استخدام البرنامج سهلاً ونلاحظ كذلك أنها تنقسم إلى عدة أقسام لكل منها عمل معين سوف نلقي الضوء عليه.

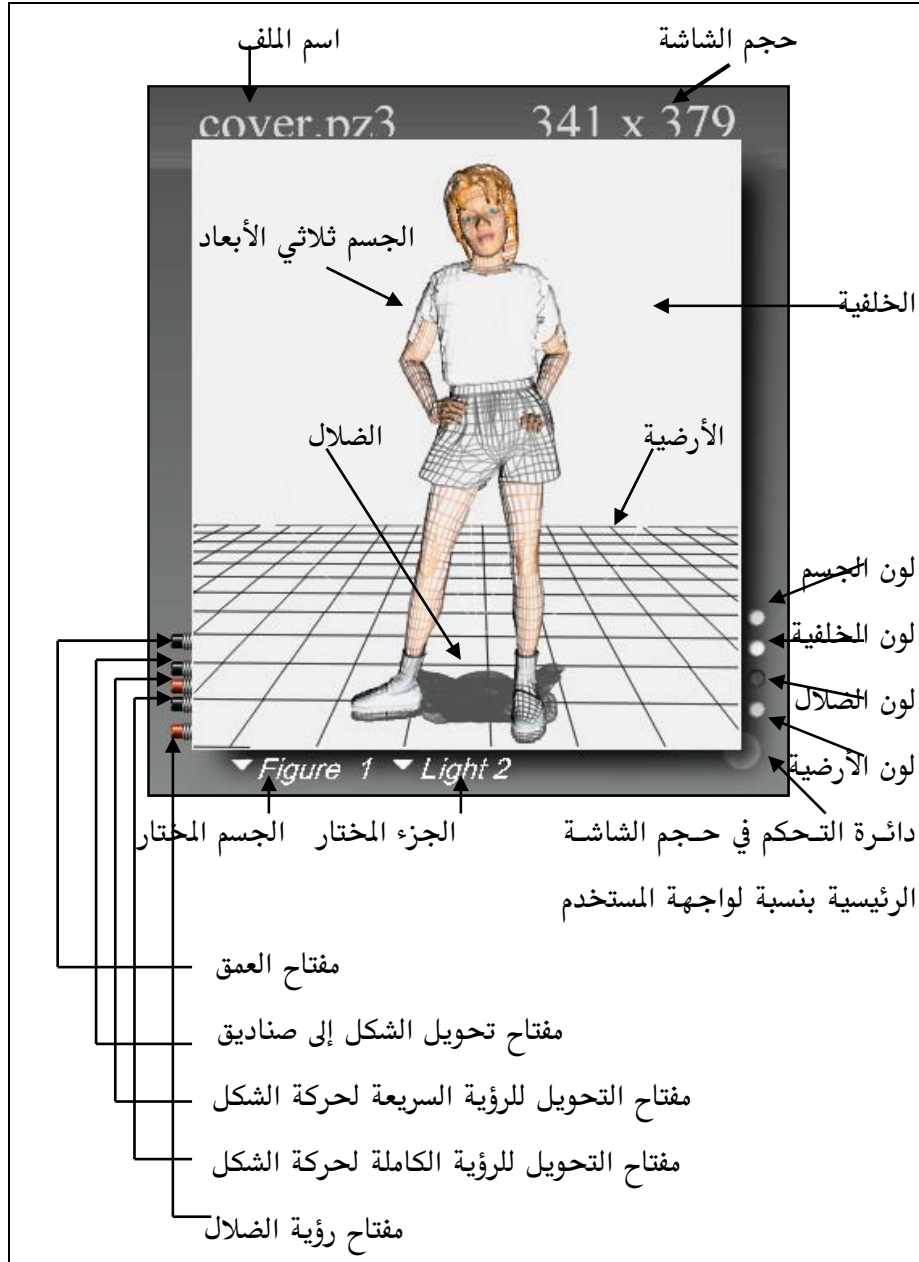
الشاشة الرئيسية

الشاشة الرئيسية هي الجزء الأساسي من واجهة البرنامج للتعرف على حالة الجسم وكذلك للتحكم به وبحركته بشكل مباشر ، من خلالها نستطيع معرفة كل ما يحدث للجسم من تغيير في الحركة والحجم. لنتعرف على أوامر الشاشة الرئيسية (الشكل في الصفحة التالية) مع بعض التوضيح:

- في أعلى اليمين نرى حجم الصورة (أو الفلم) الذي سوف ننتجه. هذا حجم الشاشة الرئيسية للجسم وهو كذلك حجم النتيجة النهائية التي سوف نحصل عليها ما لم نقوم بتغيير الحجم الناتج عن طريق الأمر **Render** (الأمر **Render** هو نفس الأمر الذي تعرفنا عليه في البرنامج السابق **Bryce 3D** ويقوم بنفس العملية) وهو المختص بإعطائنا النتيجة النهائية وسوف نتعرف عليه بشكل أوضح فيما بعد.

بعض التوضيح لحجم النتيجة النهائية: عندما نقوم باختيار الأمر **Render** من القائمة العلوية في واجهة البرنامج ، فإن **Poser3** يقوم بإعطائنا صورة معينة. سواء كانت من نوع **Bmp** أو **Tif** أو غير ذلك من الأنواع المختلفة التي يوفرها

، أو يقوم بإعطائنا فلم من النوع Avi. وللصورة أو للفلم حجم معين ، مثلا 640 x 480 أو 320 x 240 وهكذا (أيضا كما كان الحال مع برنامج Bryce 3D).



- كذلك في أعلى اليسار نرى اسم الملف الحالي (عند فتح ملف جديد يكون الاسم "UNTITLED").
- نرى الجسم ثلاثي الأبعاد في منتصف الشاشة الرئيسية (طبعا نستطيع إضافة أي عدد من الأجسام ثلاثية الأبعاد حسب إرادتنا) وكذلك الأرضية والضلال واعتقد بأن هذه العناصر الثلاث واضحة ولا تحتاج إلى تعريف أما الخلفية فهي اللون الخلفي ويمكننا استخدام لون معين أو صورة معينة أو حتى فلم متحرك.



لماذا نستخدم فلم متحرك كخلفية؟

افرض أننا نريد أن نجعل رجل يمشى على طريق ، فنقوم أولاً بإنتاج حركة الكاميرا على طريق معين في أحد البرامج ثلاثية الأبعاد كالبرنامج السابق Bryce 3D ثم تخزينه في ملف من نوع Avi على القرص الصلب ، بعد ذلك نجلبه إلى برنامجنا كخلفية متحركة ثم نقوم بإعطاء الجسم في برنامجنا حركة المشي وتعديلها مع الخلفية المتحركة فتكون النتيجة النهائية رجل يمشى على طريق.

في جهة اليمين نستطيع التحكم في الأوامر التالية:

- اللون الأمامي : وهو لون الجسم في حالة لو كان سلكي (أي يعرض بشكل سلكي)
- لون الخلفية : وهو لون خلفية المشهد
- لون الضلال : لون ضلال أي جسم ثلاثي في المشهد

- دائرة التحكم في حجم الشاشة الرئيسية بنسبة لواجهة المستخدم : عن طريق هذا الدائرة نستطيع تكبير وتصغير حجم الشاشة الرئيسية.
- أما من جهة اليسار فإننا نستطيع التحكم في الأوامر التالية :
 - مفتاح العمق : عندما نقوم بتشغيل هذا المفتاح (عن طريق الضغط عليه بإشارة الفأرة طبعاً كما هو الحال مع لأوامر الأخرى) فإننا نخبر البرنامج بوزن بان يعطي شيء من العمق للصورة بحيث يكون اللون البعيد باهت واللون القريب غامق.
 - مفتاح تحويل الشكل إلى صناديق : عن طريق هذا المفتاح نقوم بجعل الشكل عبارة عن صناديق تمثل أجزاء الجسم. ونستخدم هذا الأمر بقصد كسب بعض السرعة عند تحريك الشكل (لماذا ؟ لأن الكمبيوتر يستطيع حساب الصناديق بشكل أسرع بكثير ما لو كان الجسم في حالة أخرى).
 - مفتاح التحويل للرؤية السريعة لحركة الشكل : يقوم هذا الأمر بتحويل الشكل إلى صناديق عند اختبار الحركة وذلك حتى نستطيع أن نرى الحركة بشكل أوضح.
 - مفتاح التحويل للرؤية الكاملة لحركة الشكل : عند اختيار هذا الأمر يقوم الكمبيوتر بإعطائنا الشكل الكامل للجسم وليس مجرد صناديق ، طبعاً هذا الاختيار يحتاج إلى كمبيوتر ذو طاقة عالية.
 - مفتاح رؤية الظلال : هذا المفتاح يسمح لنا بإضافة أو إزالة الظلال من على المشهد.

- وأخيرا في اسفل الشاشة الرئيسية نستطيع التحكم في الأوامر التالية:
- الجسم المختار : نستطيع أن نختار جسم معين في المشهد عن طريق هذا الأمر أو عن طريق الضغط عليـة بإشارة الفأرة (أحيانا يكون من الصعب اختيار جسم معين عن طريق إشارة الفأرة خاصة إذا كان هناك أجسام عديدة في المشهد لذلك وضع هذا الأمر لتسهيل تلك العملية).
 - الجزء المختار : نستطيع أن نختار أي جزء من الجسم المختار عن طريق هذا الأمر أو عن طريق الضغط عليـة بإشارة الفأرة. مثلا قمنا باختيار الجسم الأول ثم نقوم باختيار الجزء "اليـد اليسار" من الجسم المختار.



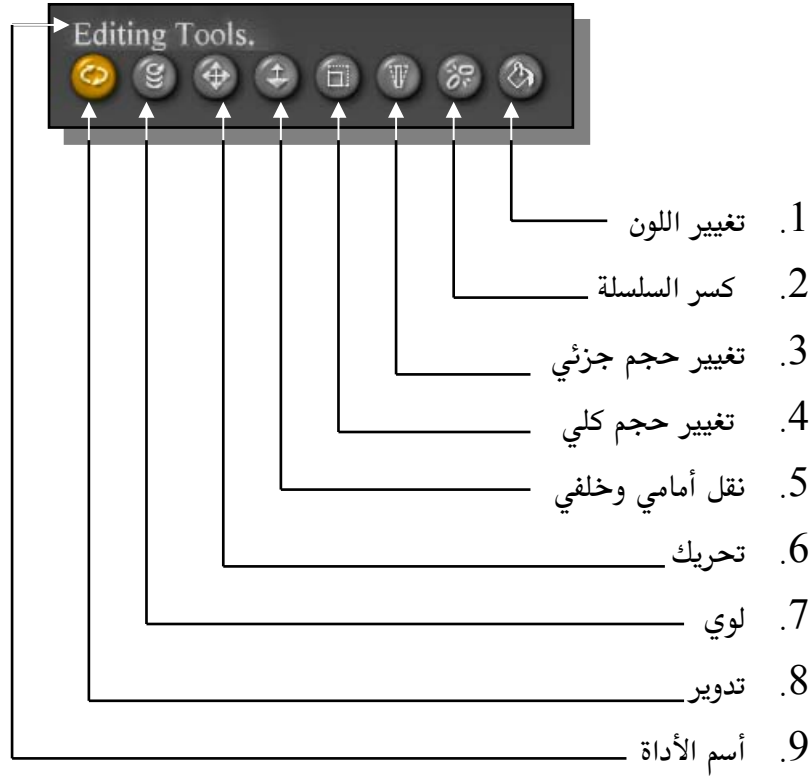
لماذا نحتاج أن نختار أي جسم ؟

نختار الأجسام أو الإضاءة من المشهد حتى نستطيع أن نقوم بتغييرها أو تعديلها حسب إرادتنا.

الآن بعد أن تعرفنا على جميع أوامر الشاشة الرئيسية نستطيع أن نقول أنها ببساطة تعطينا التحكم في ألوان و حركة وحجم المشهد حتى نستطيع القيام بعملنا المطلوب بالسهولة الممكنة وليس هناك أية صعوبة في فهم هذه الأوامر وتستطيع أن تتعرف عليها بنفسك عن طريق تجربتها بشكل مباشر على حاسبك الآلي.

أدوات وأسطوانات التحكم بالجسم

أعلى الشاشة الرئيسية مباشرة نرى أدوات التحكم بالجسم كما يلي :



- 1- تغيير اللون : نستخدم هذه الأداة لتغيير لون الجزء المختار
- 2- كسر السلسلة : الجسم الثلاثي تكون أجزائه مربوطة ببعضها البعض عن طريق سلسلة (تمثل الهيكل العظمي للجسم، مثلا اليد مربوطة بالمعصم والمعصم مربوط بالساعد والساعد مربوط بالعضد والعضد مربوط بالكنتف وهكذا) بحيث تتغير هذه السلسلة حسب حركة أي جزء، مثلا عند تغيير مكان اليد (في أول

السلسلة) سوف يتأثر الكتف (في آخر السلسلة) بذلك نستخدم هذه الأداة في الجزء المراد عنده كسر هذه السلسلة الحركية بحيث يتوقف التأثير على باقي الأجزاء.

3- تغيير حجم جزئي : نستخدم هذه الأداة لتغيير حجم الجزء المختار من أحد أطرافه بحيث يبقى الطرف الآخر بنفس الحجم.

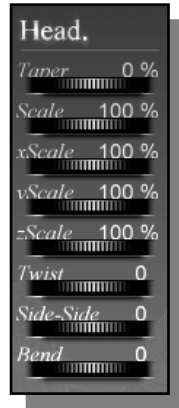
4- تغيير حجم جزئي : نستخدم هذه الأداة لتغيير حجم الجزء المختار.

5- نقل أمامي وخلفي : نستخدم هذه الأداة لنقل الجزء المختار إلى الخلف (بعيدا عن المشاهد) أو جلبه إلى الأمام (جهة المشاهد).

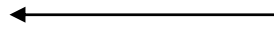
6- تحريك : هذه أداة تحريك الجزء المختار إلى أي جهة نشاء.

7- لوي أو لف : نستخدم هذه الأداة لللف الجسم المختار حيث نقوم بتدوير أحد أطرافه ويبقى الطرف الآخر كما هو.

8- تدوير : نستخدم هذه الأداة لتدوير الجسم المختار .



9- أسم الأداة : في هذه الجزء نرى اسم الأداة التي نستخدمها إذا لم نكن نستخدم أي أداة فإننا نرى الاسم "Editing Tools" مكتوب.



اسطوانات التحكم

من جهة أخرى لا تختلف أسطوانات التحكم كثيراً في عملها عن أدوات التحكم. كل ما في الأمر أننا نستطيع أن

نتحكم في درجة الأداة عن طريق تدوير الأسطوانة المطلوبة. مثلا اسطوانة التدوير أو اسطوانة تغيير الحجم. والسبب في كونها مستقلة عن أدوات التحكم أننا نستطيع التحكم في الجزء المختار بشكل أكثر دقة مثلا نستطيع تغيير الحجم

هنا على المحور السيني فقط أو الصادي فقط وهكذا. وعند اختيار جزء معين تظهر لنا أسطوانات التحكم المتعلقة به.

النقاط

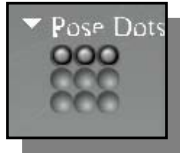
هناك ثلاث أنواع من النقاط في برنامج Poser 3 كما يلي:

1- نقاط التحكم بالإضاءة:



وهي عبارة عن ثلاث كرات تمثل كل منها إضاءة في المشهد لتعطي مجموع ثلاث إضاءات. ونتحكم بها عن طريق تحريكها أو تغيير لونها.

2- نقاط الذاكرة:



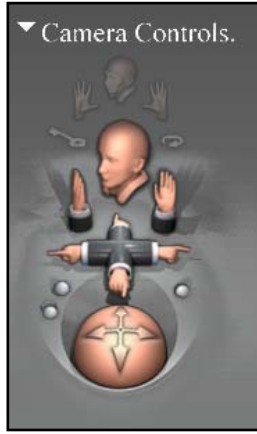
نستخدم هذه النقاط قبل القيام بتغيير معين سواء في المشهد أو لجزء معين بحيث تحفظ في ذاكرة الكمبيوتر كل ما في المشهد وذلك في اللحظة التي نضغط فيها على نقطة معينة ثم بعد ذلك نستطيع أن نعود إليها عن طريق الضغط عليها مرة أخرى.

3- نقاط التحكم في شكل الجسم على الشاشة الرئيسية:



نستخدم هذه النقاط لنتحكم في شكل الجسم الثلاثي الأبعاد في الشاشة الرئيسية سواء كان النوع السلبي أو الكرتوني أو إلى آخر ذلك من تسعة أنواع مختلفة.

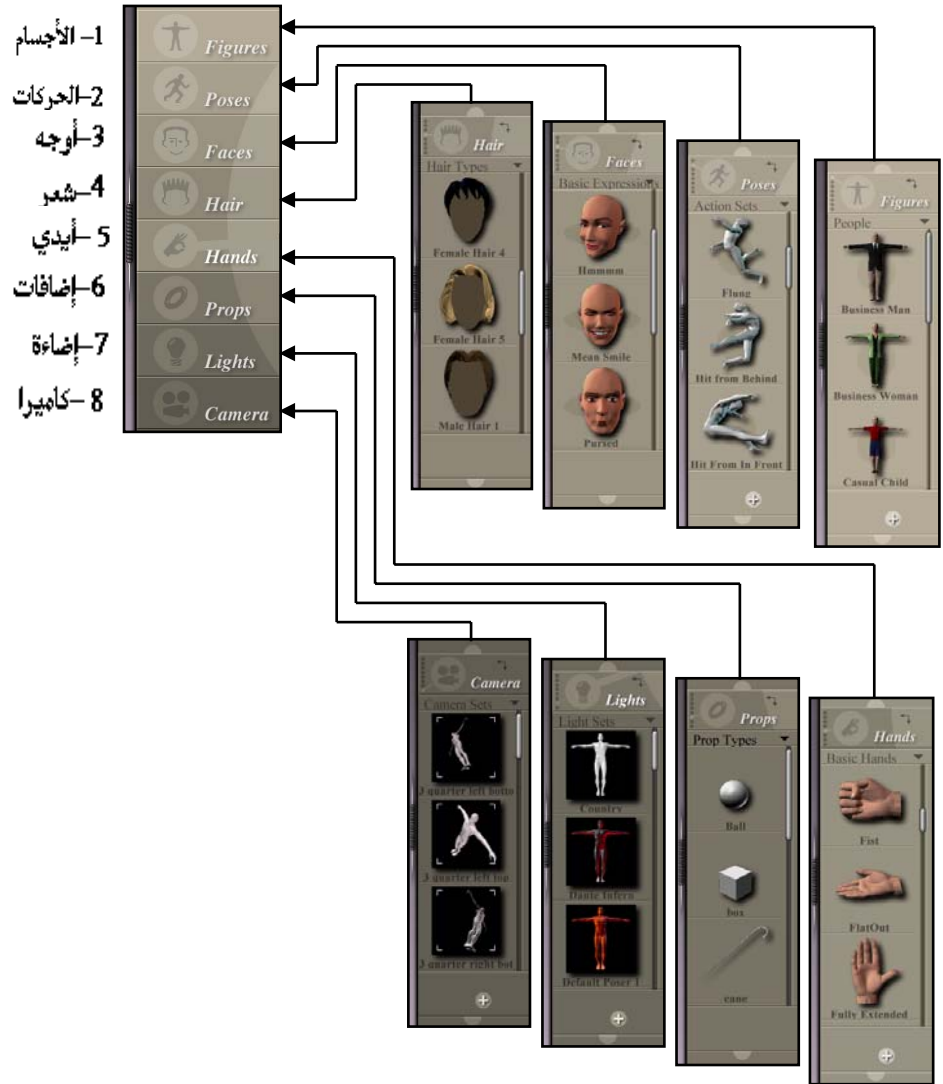
أدوات التحكم بالكاميرا



عن طريق أدوات التحكم بالكاميرا نستطيع أن نتحكم في حركة الكاميرا بالشكل الذي نريد. نحن نعلم بان كل ما نراه في المشهد هو عن طريق الكاميرا وسوف نحتاج إلى هذه الحركة بحرية للقيام بالتغييرات المناسبة في المشهد لكي تكون الأمور بالشكل المطلوب. وكل ما علينا فعله هو الضغط على أحد الأشكال الظاهرة في الصورة على اليسار وتقوم الكاميرا بالحركة

حسب الشكل. والجدير بالذكر هو أن ما نراه في أعلى أدوات التحكم وهو الرأس فعن طريق الضغط عليه تتحول الكاميرا مباشرة إلى وجه الجسم والحكمة في ذلك أننا نحتاج في أوقات كثيرة إلى التحكم في ملامح الوجهة وتوفر علينا هذه الطريقة الوصول إلى الوجهة بشكل سريع.

مستطيل المكتبات

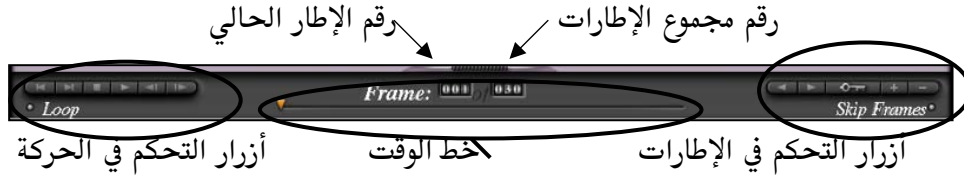


مستطيل المكتبات هو قلب برنامج Poser 3 ويوفر علينا الكثير من الوقت. وهو

- عبارة عن مكتبات جاهزة تقوم بما يلي :
- 1- الأجسام : عن طريق هذه المكتبة نستطيع أن نختار أجسام مختلفة من (حيوانات ، أطفال ، شيوخ ، نساء ، وكذلك ملابس مختلفة)
 - 2- حركات : هناك الكثير من الحركات سواء الثابتة أو المتحركة يمكن استخدامها عن طريق هذه المكتبة مع أي جسم في المشهد.
 - 3- أوجه : في هذه المكتبة نختار أوجه مختلفة من وجه ضاحك ، جزين ، أو ناطق بحرف معين ونستطيع طبعا أن نستخدم الحركة لنقل الوجه من تعبير إلى آخر (مثلا لو أردنا أن نجعل رجل يتكلم ثم يضحك).
 - 4- شعر : مكتبة الشعر تحتوي على أنواع مختلفة من الشعر ، شعر طويل ، قصير ، شعر رجل ، أو شعر فتاة .. الخ .
 - 5- أيدي : مكتبة الأيدي تحتوي على الكثير من حركات لليد التي قد تحتاجها مع الكثير من التفاصيل.
 - 6- إضافات : عن طريق هذه المكتبة نستطيع أن نضيف عوامل أخرى للجسم. مثلا. قبعة رأس أو عصا .. الخ
 - 7- الكاميرا : وأخيرا تحتوي مكتبة الكاميرا على أوضاع مختلفة للكاميرا نستطيع استخدامها كما نريد.
- توفر علينا المكتبات في هذا البرنامج الكثير من الوقت للوصول إلى النتيجة المطلوبة بدون الحاجة إلى إعادة العمل مرات عديدة ونستطيع أن نضيف إلى هذه المكتبات كل ما نقوم به بأنفسنا من حركة الكاميرا وأجسام ثلاثية.

الحركة

مستطيل التحكم في الحركة :

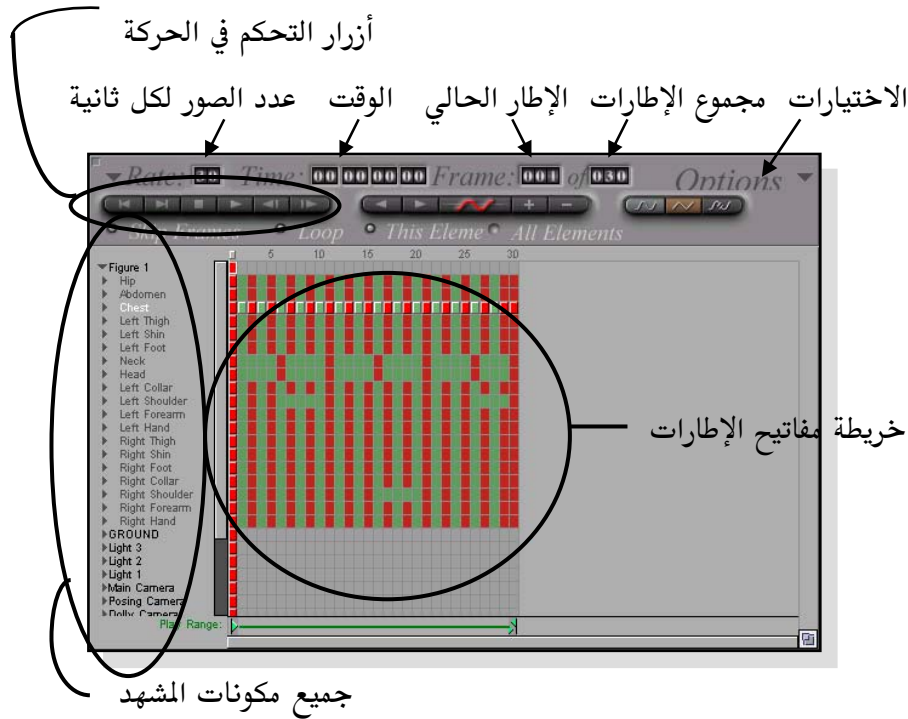


مستطيل التحكم في الحركة هو نافذة الحركة لهذا البرنامج. و عن طريقة نستطيع أن نتحكم في حركة كل شيء في المشهد ، من إضاءة وحركة الأجزاء المختلفة للجسم. ونستطيع كذلك أن نحفظ هذه الحركة في مكتبة الحركات إذا أردنا. وعلى يسار مستطيل الحركة نرى أزرار التحكم في الحركة ، وتبدو كأنها أزرار مسجل الفيديو وفي الحقيقة أنها تقوم بنفس العمل. أما في المنتصف فإننا نرى مربع يدل على رقم الإطار الحالي للحركة ومربع آخر يدل على رقم مجموع الإطارات في المشهد ويتمثل الوقت بخط في منتصف مستطيل الحركة أما موقع الإطار الحالي من الوقت فيعبر عنه بسهم متجه إلى الأسفل.

أزرار التحكم في الإطارات :

عن طريق هذه الأزرار نستطيع أن نتحكم في إطارات الحركة للمشهد ، ونحن نعلم بان أي حركة سينمائية تحدث على فترة معينة نعبّر عنها بالإطارات. مثلا في النظام الأمريكي NTSC هناك ثلاثون إطار في الثانية أو بمعنى آخر نحن نرى

ثلاثين صورة مختلفة لشيء معين خلال كل ثانية فتبدو لنا الحركة. وهذا يعني أن كل إطار هو عبارة عن صورة مختلفة قليلا عن الصورة أو الإطار الذي يسبقها فتبدو لنا الحركة. ولو نظرنا إلى أزرار الإطارات في الشكل السابق لرثينا أن هناك زر علية صورة مفتاح ، وبمجرد الضغط على هذا الزر تظهر لنا شاشة الحركة كما يلي:



من خلال شاشة الحركة نستطيع وبشكل دقيق التحكم في مراقبة كل حركة من مكونات المشهد (مثل الإضاءة وحركة الأجسام الثلاثية). فعن طريق خريطة مفاتيح الإطارات نستطيع أن نتعرف على موقع كل مفتاح (مفاتيح الحركة أو تعرف كذلك بمفاتيح الإطارات ما هي إلا إطارات يحدث عندها تغير غير طبيعي للحركة) لحركة كل شيء في المشهد فاللون الأحمر (أو اللون الغامق

إذا لم يطبع هذا الكتاب بالألوان) يدل على مفتاح الحركة لجسم معين. وكما نرى بأن أسماء مكونات المشهد تظهر على جهة اليسار من شاشة الحركة وعلى يمينها نرى خريطة مفاتيح الإطارات لكل منها.

القائمة العلوية

File Edit Figure Display Render Animation Window Help

كما هي العادة في أغلب برامج النظام ويندوز فإننا نقوم عن طريق القائمة العلوية بالقيام ببعض الأوامر الخاصة بالبرنامج والتي تعودنا على وجودها في القائمة العلوية (تخزين الملفات ، فتح الملفات الخ).

الخلاصة

لقد حاولنا أن ننظر إلى أهم أوامر هذا البرنامج وأعتقد أننا أطلعنا عليه بما فيه الكفاية ، وهناك طبعاً الكثير مما لم نلق عليه النظر ولا نستطيع أن نتكلم عن كل تفاصيل هذا البرنامج وإلا فإننا سنخرج عن موضوعنا الأصلي في هذا الكتاب وهو برمجة الصوت والصورة والألعاب على الكمبيوتر. ولكن بمعلومية ما سبق وشرحناه اعتقد أنك عزيزي القارئ سوف تستطيع استخدام هذا البرنامج بدون الحاجة للنظر إلى الكتب الملحقه به عند شرائه. برنامج بوزر من الإضافات المهمة لكل مبرمج للصوت والصورة على الكمبيوتر. واعتقد أن كل من يستعمل هذه النوعية من البرامج يقدر الوقت والجهد الذي يوفره هذا البرنامج في إنتاج حركة حقيقية لشخصيات هي من صنع الكمبيوتر. حقيقة أن المبرمج بإمكانه القيام برسم هذه الأشكال واستخدامها مباشرة في البرنامج ولكن لا يمكن أن تكون هذه الأشكال بمستوى الأجسام ذات الأبعاد الثلاثية والتي هي من إنتاج الكمبيوتر. في هذا الكتاب سوف نتعرف على برمجة الأبعاد الثلاثية. وعن طريق هذه البرامج سوف نحتاج إلى شخصيات ذات ثلاث أبعاد. عندها سوف نرى ضرورة استخدام هذا البرنامج وغيرها من برامج الأبعاد الثلاثية لإنتاج الأجسام المطلوبة. ولإعطائك فكرة عن هذا البرنامج سوف أطلب منك أن تذهب إلى أحد محلات الألعاب لترى عدد برامج الألعاب التي تستخدم الأجسام الثلاثية الأبعاد وكذلك برامج التصور الحقيقي وبرامج الطيران وكلها أصبحت تستلزم هذه النوعية من البرامج. ليس طبعاً هذا البرنامج الوحيد في الأسواق الذي يقوم بهذه العملية ولكن عند وضع الثمن ، القدرة ، والسرعة في الاعتبار فإن هذا البرنامج هو ما تبحث عنه.

الفصل العاشر

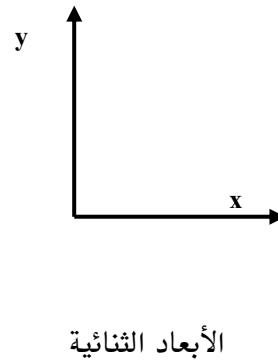
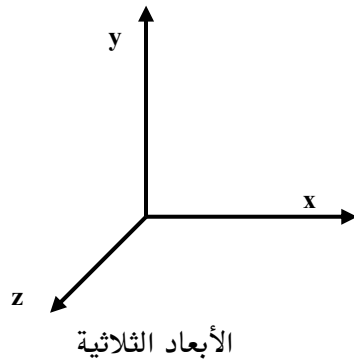


البرية لعام ثلاثي الأبعاد



2D & 3D

عزيزي القارئ بعد أن تعرفنا على كيفية البرمجة بالأبعاد الثنائية وكيفية التعامل مع الشاشة على أساس المحورين السيني (x) والصادي (y) وكذلك استخدمنا بعض البرامج ثلاثية الأبعاد لإنتاج بعض الرسوم التي استخدمناها في الأمثلة السابقة ، وذلك لإعطاء الإحساس أن المشهد ثلاثي الأبعاد وإعطاء الممثلين نوع من العمق ، نأتي إلى الحقيقة أننا لازلنا في عالم ثنائي الأبعاد ، وأننا نتعامل مع المشهد على أساس (x) و (y) فقط. ويأتي السؤال الملح الآن لكل من يتعامل مع برمجة الصوت والصورة على الكمبيوتر وهو "كيف نستطيع أن نبرمج الأبعاد الثلاثية؟" وأطلقنا عليها الأبعاد الثلاثية لأننا سوف نضيف بعد ثالث وهو البعد (z) بالإضافة طبعا إلى البعدين (x) و (y).



عندما نتعامل مع الأبعاد الثلاثية فإننا نتعامل مع صور سطحية ذات بعدين وكل ما نقوم به هو تنسيق بين تلك الصور ثنائية الأبعاد على الشاشة وبين الصوت ونوع الإدخال المطلوب. أما في حالة الأبعاد الثلاثية فالأمور أكثر واقعية ، فنحن لا نتعامل مع مجرد صور وأصوات لها تناسق معين وإنما نتعامل مع أجسام ثلاثية الأبعاد تتأثر بالعوامل الطبيعية مثل الجاذبية والإضاءة ولها حجم معين في الفضاء. وقبل أن نتعمق في شرح برمجة الأبعاد الثلاثية قد تتساءل عزيزي القارئ عن مدى صعوبة وفعالية الأمثلة التي سوف نتطرق إليها في هذا الكتاب ، ولإجابة هذا التساؤل يجب أن نعرف بعض الحقائق عن هذه النوعية من البرامج.



سنأخذ على سبيل المثال المكتبة المستخدمة لإنتاج لعبة ثلاثية الأبعاد شهيرة مثل لعبة "Blood" (تستطيع الحصول على نسخة تجريبية ومعلومات أكثر عن هذه اللعبة من

موقع الشركة المصنعة على الإنترنت <http://www.the-chosen.com> وتسمى "DirectEngine" تلك المكتبة تكلف آلاف الدولارات (بالضبط تكلف \$250000 ، لا هذا ليس خطأ مطبعي إنما فعلا تكلف مائتان وخمسون ألف دولار أمريكي لكل برنامج تصممه باستخدامها وتستطيع أن تحصل على معلومات أكثر من هذا الموقع <http://www.lithtech.com>) ولهذا السبب قرر الكثير من المبرمجين كتابة مكتباتهم الخاصة وهناك الكثير المتوفر منها مجانا على الإنترنت. ولكن لو رجعنا إلى الأمر الواقع وهو أن جميع هذه المكتبات المجانية لا تصل إلي المستوى المطلوب. وقد كنت أفكر كثيرا عند كتابتي لهذا الكتاب عن كيفية شرح هذه المشكلة للقارئ الكريم ولم أستطع تصور خيبة الأمل التي سوف يواجهها عند معرفته بذلك. وحتى

لو أننا قمنا بكتابة مكتبة خاصة لهذا الكتاب تتعامل مع الأبعاد الثلاثية ستكون محدودة المستوى، ولن نستطيع استخدامها فعلياً لكتابه برامج ذات فائدة تتعادل مع مستوى البرامج التجارية، إلى أن (لاحظ أن هذه المشكلة فقط مع برمجة الأبعاد الثلاثية ولكن في برمجة الأبعاد الثنائية فإن مكتبة AGDX توفى بالغرض على أكمل وجه ونحن في البرمجة العربية <http://www.ArabGames.com> نستعملها فعلا بشكل تجاري).

.... إلى أن أتى أخيراً الحل الذي لم يخطر لأحد على بال.

قامت شركة "Eclipse Entertainment" بعرض مكتبتها "Genesis 3D" والتي "تتفوق" على الكثير من المكتبات التجارية الأخرى للبيع - كما هو الحال مع هذه النوعية من المكتبات التجارية- ولكن بعد ذلك بفترة بسيطة عند كتابة هذا الكتاب قامت هذه الشركة بخطوة غير مسبوقه وذلك بعرض مكتبتهم "مجانا" (نعم هذا يعني انك تستطيع استخدامها بدون أن تدفع شيء، انظر إلى النص المأخوذ من ترخيص الاستعمال أدناه) للاستعمال الشخصي أو التجاري. وبشرط واحد فقط وهو عرض علامتهم التجارية عند تشغيل البرنامج.

وهنا تتساءل عزيزي القارئ هل هذه المكتبة فعلا أفضل من المكتبات التجارية الأخرى والتي تباع بآلاف الدولارات؟ والإجابة ببساطة "نعم" بنظرة واحدة إلى المثال الملحق مع هذا الكتاب (بعد تركيب المكتبة من القرص المدمج شغل المثال Gtest.exe) وسوف يعطيك الدليل على هذا القول.

The Genesis3D SDK is free. We built it for ourselves and now we're giving it to you. There is absolutely no charge to download the SDK, build a product with it, and sell it. There is no royalty to pay. We are making 3D free for everyone. The only requirement for any product built with the SDK is the Genesis3D logo must be displayed unmodified on startup. Show our logo. That's all we ask.

ترجمة النص أعلاه :

"المجموعة البرمجية لمكتبة Genesis3D مجانية. نحن بنيناها لأنفسنا والآن نعطيهما لك. ليس هناك أي تكاليف مقابل استعمال المكتبة على الإطلاق ، و بناء برامجك بها ، ثم بيع تلك البرامج. وليس عليك أن تدفع لنا مقابل ذلك أي حقوق. نحن نجعل من برمجة 3D مجاناً للجميع. الشيء الوحيد الذي نطلبه لأي برنامج يستغل هذه المكتبة هو وجوب وضع علامة لـ Genesis3D لتظهر بدون أي تعديل عند بداية البرنامج. ظهور علامتنا هو كل ما نطلبه."

هذا النص مأخوذ من ترخيص الاستعمال للمكتبة

سوف نقوم في هذا الكتاب باستخدام هذه المكتبة في إنتاج الكثير من الأمثلة والتي توفر لنا القدرة على تصميم ما يحلو لنا من برامج الأبعاد الثلاثية ، سواء للاستعمال التعليمي ، الترفيهي أو حتى التدريبي.

❖ ما هي علاقة 3D Genesis بتكنولوجيا DirectX وخاصة العنصر Direct3D لأنه المهتم بالإبعاد الثلاثية؟

سوف نتطرق بشيء من العمق إلى برمجة العنصر Direct3D في الجزء الثاني من هذا الكتاب ولكننا في هذا الفصل سوف نستخدم المكتبة Genesis 3D والتي بدورها تقوم باستغلال ذلك العنصر بالإضافة إلى استغلال سرعات 3DFX (المشهورة بـ Voodoo والتي هي أيضا تستخدم تكنولوجيا برمجية خاصة بها يطلق عليها Glide

وهذه شبيبهه بتكنولوجيا (DirectX) وكذلك تكنولوجيا MMX (MultiMedia eXtension) إذا توفرت في المعالج الرئيسي ثم تكنولوجيا 3D-Now والموجودة حاليا في المعالجات من شركة AMD وأخيرا تقوم باستغلال قوة المعالج الرئيسي (طبعا ذلك في أسوء الحالات عند عدم وجود أي نوع من السرعات على الجهاز المستخدم). كل ذلك وبدون إضافة أي نوع من البرمجة الخاصة إنما على العكس أصبحت البرمجة باستخدام هذه المكتبة أكثر سهولة ومتعة (صدقني لقد كنت مبتعدا عن برمجة الأبعاد الثلاثية حتى رأيت هذه المكتبة وسوف نتعرف عليها معا من خلال الأمثلة).

مميزات مكتبة Genesis 3D

- سرعة عالية في العرض
- استخدام المرايا (الأسطح العاكسة كالمرايا أو مستنقع أو ... الخ)
- الضلال المتحركة
- استخدام الضباب
- الأشكال الخارجية (Texture) المتحركة
- الحركة الموجية للسطح في الوقت الحقيقي (تخيل مشاهدة بحر مثلا)
- خمس أنواع مختلفة من الإضاءة
- تغيير الأشكال في وقت حقيقي (Morphing Vertex)
- استخدام الممثلين ثلاثي الأبعاد وبحركة واقعية في وقت حقيقي.
- العلاقة الحركية بين الممثلين.

- استخدام ملفات برنامج 3D-Studio Max
- استخدام الأشكال الخارجية الشفافة (لسطح زجاجي مثلاً)
- تغيير الأشكال الخارجية في وقت حقيقي (مورفين Texture morphing)
- درجة وضوح الشاشة (حتى نستطيع زيادة السرعة على الأجهزة القديمة)
- تغيير الممثلين في وقت حقيقي (مورفين Character morphing)
- استغلال سرعات DirectX أو 3DFX أو فقط المعالج الرئيسي
- الاستفادة من تكنولوجيا MMX أن وجدت ولكن مع عدم الحاجة لها.
- الاستفادة من تكنولوجيا 3D NOW أن وجدت للمعالجات من شركة AMD.
- سهوله الأوامر البرمجية
- برنامج تصميم العوالم ثلاثية الأبعاد "المصمم" World Editor
- استخدام القوانين الفيزيائية (مثل الجاذبية ، التوازن ، الرياح ... الخ)
- استخدام برمجة شبكات الاتصال لأكثر من مستخدم أو لاعب (عن طريق الإنترنت مثلاً).
- الميزة الأساسية وهي أن هذه المكتبة بكل هذه المميزات مجانية للاستخدام الخاص والتجاري.

خطوات إنشاء برنامج عن طريق مكتبة Genesis 3D

قبل الدخول في التفاصيل البرمجية والتي كانت تخيف الكثيرين (بما فيهم أنا) من الدخول في عالم برمجة الأبعاد الثلاثية، دعونا نتأكد بأن الهدف من هذه المكتبة هو وضع ما كان يعرف بالصعب إلى عمل سهل نستطيع أن نتعامل معه (كما هو الحال مع مكتبة AGDX). وإذا كنت متابع معنا برمجة الأبعاد الثنائية واستخدام مكتبة AGDX فلن تجد الكثير من الاختلاف في طريقة التعامل مع هذه المكتبة وسوف نقوم تقريبا بنفس الأسلوب البرمجي الذي اتبعناه من قبل. عند كتابة برنامج ثلاثي الأبعاد يجب أن نتبع الخطوات الثلاث التالية:

1- نستخدم برنامج إنشاء عوالم الأبعاد الثلاثية **World Editor** (هذا البرنامج يأتي مجانا مع القرص المدمج وتحصل عليه من ضمن برامج مكتبة Genesis 3D) في إنتاج العالم الثلاثي المطلوب. هذه الخطوة ليس فيها أي برمجة وكل ما نقوم به استخدام الأدوات المتوفرة عن طريق هذا البرنامج في تصميم وإنتاج عالم ثلاثي الأبعاد.

2- نقوم بحفظ الملفات الناتجة على القرص الصلب (كما فعلنا عند إنتاج خلفيات التايلز في برنامج **MapMaker** والملفات الناتجة لاستخدامها مع المكتبة AGDX).

3- نكتب شفرة برنامجنا والذي يقوم باستخدام ذلك العالم الذي صنعناه عن طريق برنامج إنشاء عوالم الأبعاد الثلاثية (أو للسهولة سوف نسميه "المصمم")

4- إنشاء المثلين ثلاثي الأبعاد (نستطيع أن نستخدم الكثير من برامج الأبعاد الثلاثية في هذه الخطوة بما في ذلك برنامج 3D-Studio Max والذي نستطيع من خلاله كذلك جلب الحركة للممثل وبرامج أخرى مثل TrueSpace)

تركيب مكتبة Genesis3D

النسخة الموجودة على القرص المدمج الملحق بالكتاب هي نسخة كاملة لمكتبة Genesis3D مع كامل الملحقات التابعة لها وتأتي كذلك مع الشفرة البرمجية التابعة لها (هناك شروط خاصة يجب الإطلاع عليها إذا أحببت تغيير الشفرة البرمجية). سوف نقوم بشرح الخطوات اللازمة لتركيب هذه المكتبة على جهاز الحاسب الآلي حتى يتسنى لنا استخدامها :

❖ بعد وضع القرص المدمج في الكتاب سنرى القائمة التلقائية (ملاحظة : إذا لم تظهر القائمة بشكل تلقائي شغل الملف autorun.exe مباشرة من على القرص المدمج) :



من القائمة التلقائية للقرص المدمج الملحق بالكتاب نختار "صفحة التركيب".
وهي الصفحة المسؤولة عن تركيب أهم البرامج والأدوات لهذا الكتاب.

❖ بعد الضغط على صفحة التركيب سنرى النافذة التالية :



وكما نلاحظ أننا عن طريق هذه النافذة نستطيع تركيب المكتبتين AGDX و Genesis 3D بالإضافة لتكنولوجيا DirectX وكذلك البرنامج Bryce 3D. هناك كذلك برامج أخرى موجودة على القرص المدمج نستطيع تركيبها ولكن هذه هي أهم البرامج لهذا الكتاب.

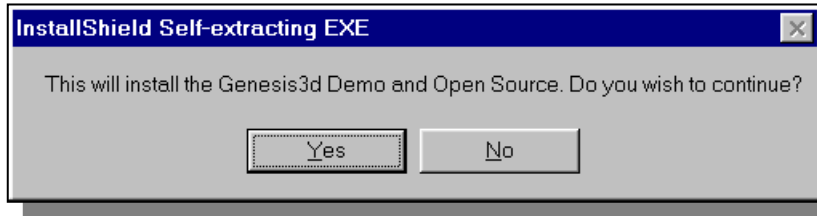
❖ من القائمة نختار "تركيب تكنولوجيا DirectX".

سوف نرى الشاشة التالية :



❖ من هذه النافذة كل ما علينا فعله هو الضغط على "زر التركيب".

1- سوف ترى الرسالة التالية على الشاشة :



لا تعبئنا بكلمة Demo (نسخة تجريبية) لان هذه هي النسخة الكاملة ولكن وضعت هذه الكلمة هنا حتى تخبر الشركة المصنعة لهذه المكتبة المبرمجين أن الإمكانيات النهائية للمكتبة لازالت قيد التطوير.

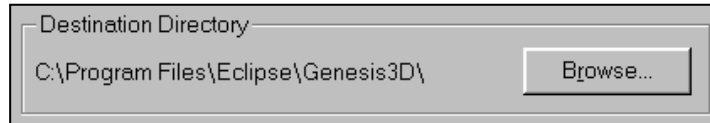
طبعا تضغط على المفتاح Yes

2 - بعد ذلك سيبدأ برنامج التركيب:

❖ اضغط على الأمر Next حتى تذهب للشاشة التالية.

❖ اضغط على الأمر Yes حتى تقبل بشروط الشركة (يفضل قراءة هذه الشروط)

5 - بعد ذلك سترى الشاشة التالية وتُسأل عن المكان الذي تود أن يقوم برنامج التركيب بوضع المكتبة وملحقاتها فيه ويعطيك الاقتراح التالي:



وكما نرى (C:\Program Files\Eclipse\Genesis\) هو المكان الذي يقترحه برنامج التركيب على القرص الصلب. إذا كان هذا هو المكان المناسب ، كل ما عليك فعله هو الضغط على الأمر Next أما إذا أردت تغيير المكان فأضغط على الأمر Browse وأختار المكان المناسب (سوف تظهر لك رسالة تسألك إذا كنت متأكد من أنك تريد إنشاء ملف مختلف ، طبعاً عندها اضغط على الأمر Yes) ثم بعد ذلك اضغط على الأمر Next.

3 - بعد أن يتم تركيب المكتبة على جهازك سوف يقوم بسؤالك فيما إذا كنت تريد قراءة الملف الملحق اضغط الأمر No (تستطيع قراءته في وقت لاحق) ثم اضغط الأمر Ok.

4- قم بإغلاق جميع النوافذ المفتوحة (عن طريق الضغط على العلامة X في أعلى يمين كل نافذة).

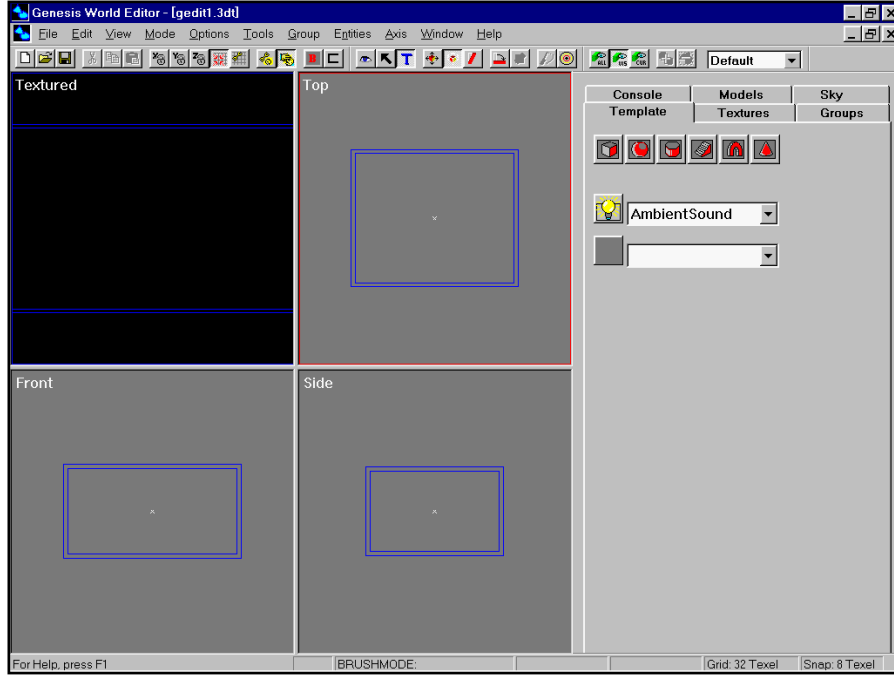
إذا نفذت الأوامر السابقة بالطريقة الصحيحة فإن برنامج التركيب قد قام بتركيب ما يلي على جهازك :

- ❖ كل الملفات البرمجية المطلوبة لإنتاج برامج تستغل هذه المكتبة.
- ❖ البرنامج World Editor وهو برنامج نقوم عن طريقة بإنشاء العوالم ثلاثية الأبعاد والتي سوف نستخدمها في برامجنا.
- ❖ المراجع التابعة له.
- ❖ المراجع التابع للمكتبة.
- ❖ البرنامج Actor Studio والذي عن طريقة سنقوم بإنشاء وتغيير الممثلين في برامجنا.
- ❖ البرنامج Actor Viewer والذي نستطيع عن طريقة رؤية واستعراض الممثلين.
- ❖ اللعبة Gtest وهي لعبة تقوم بإعطائك فكرة عن إمكانيات هذه المكتبة الثلاثية الأبعاد. وطبعاً تأتي مع الشفرة البرمجية التابعة لها.
- ❖ بعض الملاحظات عن هذه المكتبة.

وتستطيع طبعاً أن تجد كل هذه البرامج عن طريق قائمة البداية (Start) – (Genesis3D) – (Programs).

المصمم – برنامج تصميم عوالم الأبعاد الثلاثية World Editor

من قائمة البداية "Start" نختار "Programs" ثم نختار "Geness3D"
ثم نختار "World Editor"







برنامج المصمم شبيه ببرامج إنشاء الأبعاد الثلاثية الكثيرة في الأسواق , عن طريقة سنقوم بإنشاء عوالم الأبعاد الثلاثية التي نستخدمها في برامجنا. ولو سبق ومر عليك مصطلح التصور الحقيقي "Virtual Reality" وهو مصطلح يعني إنشاء عالم مشابه للعالم الذي نعيش فيه ويحتوي على اغلب صفاته. مثلا نحن في

الحقيقة تتأثر بالجاذبية كذلك نحن نتأثر بالإضاءة التي حولنا كذلك نحن نعيش في فضاء له أكثر من بعد , وكما في العالم الحقيقي نحن في التصور الحقيقي نحاول إنشاء هذا العالم ووضعه في جهاز الكمبيوتر. في برنامجنا "المصمم" سوف نقوم بنفس العملية , لن نقوم فقط بإنشاء التصور الحقيقي ولكن أفضل من ذلك ربما تستطيع أن تقول أننا سوف نأخذ مصطلح التصور الحقيقي إلى مرحلة أكثر تطورا وواقعية. في الحقيقة من خلال هذه المكتبة نستطيع استخدام عواملنا ليس في البرامج التي ننشئها فقط , سواء كانت العاب أو علمية ولكن نستطيع أن نعيشها مع الآخرين من خلال الإنترنت.

سنحاول أولا التعرف على الأوامر الرئيسية لهذا البرنامج. مثلا ماذا نعني بفرش (جمع فرشة كما في فرشة الرسم) -Brushes- كيف ننشئها وكيف نستفيد منها. كيف يمكننا أن نعطي الأجسام في عالمنا والفرش كذلك شكلها الخارجي "textures" ثم سنتعرف على entities (هذه الكلمة مثال جميل للصعوبات التي لاقيتها لإعطاء المعنى العربي المتوافق مع الكلمات الإنجليزية. فهذه الكلمة غريبة لكننا سوف نقوم بشرحها ثم نعطيها مصطلح يكون قريب للمعنى المقصود) وكيف نضيفها إلى خارطة العالم ثم أخيرا سنتعرف على كيفية إنتاج الملفات النهائية التي سنحتاج لاستخدامها في برنامجنا.

أولا يجب أن نعرف أننا عندما نعمل مع المصمم فإننا سوف نتعامل مع ثلاث حالات :

- حالة الكاميرا – Camera Mode 
- حالة الاختيار – Selection Mode 
- حالة التعديل – Modify Mode 


في رأي أن افضل طريقة للتعرف على أي برنامج للحاسب الآلي هي أن نقدم مثال معين عن طريقة. وهنا سنقوم باستخدامه لإنشاء عالم ثلاثي بسيط . في البداية سوف نختار حالة التعديل عن طريق الضغط على الزر  (عادة تكون هذه الحالة تلقائية عند بداية البرنامج) بعد ذلك سنقوم بإنشاء صندوق خالي والذي سيكون عبارة عن غرفة. الآن لنرى الأشكال التالية (على يمين الشاشة) تحت العنوان

: Template

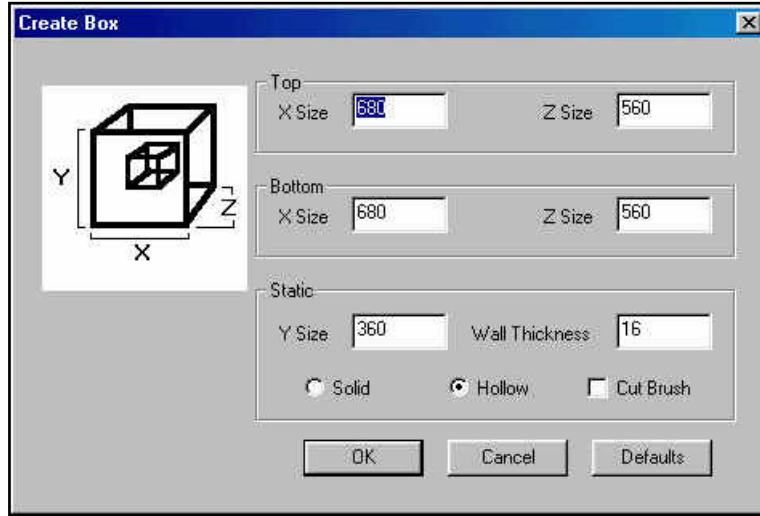


وكما نرى أن هناك اكثر من شكل. وكل هذه الأشكال يطلق عليها فرش فالصندوق فرشاة والسلم فرشاة والهرم فرشاة ونقوم عن طريق اختيار الفرشاة المناسبة بإنشاء عالمنا.

في هذا المثال سوف نقوم باختيار الصندوق وذلك

عن طريق الضغط على الفرشاة .

بعدها مباشرة سوف نرى الشاشة التالية :




وكما نرى أن هناك أكثر من مكان نستطيع تغيير المعطيات فيه (عرض , حجم , نوع الفرشاة , متانة الأوجه... الخ) في مثالنا هذا سنقوم بقبول القيم المعطاة تلقائياً من البرنامج.

لاحظ أن نوع الصندوق الخالي من الدخل "Hollow" هو النوع المختار تلقائياً لنا في هذه الفرشاة. وهذه النوعية ممتازة لإنشاء الغرف والصالات أو كل جسم هندسي خالي من الداخل. النوع الثاني وهو Solid هو لإنشاء صندوق عكس الأول أي انه ليس خالي من الداخل. النوع الآخر وهو Cut Brush مفيد جداً عندما نريد أن نقص فتحة في غرفة لإنشاء باب مثلاً أو نافذة.

لأن لنضغط على الزر "OK" لنخبر البرنامج على موافقتنا على المعطيات الموجودة.

بعدها سنعود للشاشة الرئيسية , هناك نرى مربع أو في الحقيقة مجسم لصندوق نراه من عدة جهات (من الأعلى واليسار و الجانب). ونستطيع أن نغير حجمه عن طريق الضغط بإشارة الفأرة على أحد زواياه.

الآن بعد أن أصبح لدينا مربع خالي أو غرفة لننتقل إلى للحالة الثانية للبرنامج وهي حالة الكاميرا عن طريق الضغط على  .

في هذه الحالة ببساطة نتحكم في الرؤية فنستطيع مثلا عن طريق الضغط على المفتاح اليمين لإشارة الفأرة بالاقتراب أو الابتعاد عن الجسم و المفتاح اليسار لكي نحرك موقعنا بالنسبة للجسم. كما يمكننا كذلك استخدام إشارة + الجمع أو الطرح - من لوحة المفاتيح للاقتراب أو الابتعاد.


هناك ملاحظة جيدة وهي أننا لنقوم فقط بما سبق لا نحتاج الذهاب إلى حالة الكاميرا بل يمكننا القيام بها ونحن في حالة التعديل عن طريق الضغط على مفتاح المسافة ثم استخدام مفاتيح الفأرة كما سبق أو كذلك استخدام إشارة + الجمع أو الطرح - من لوحة المفاتيح.

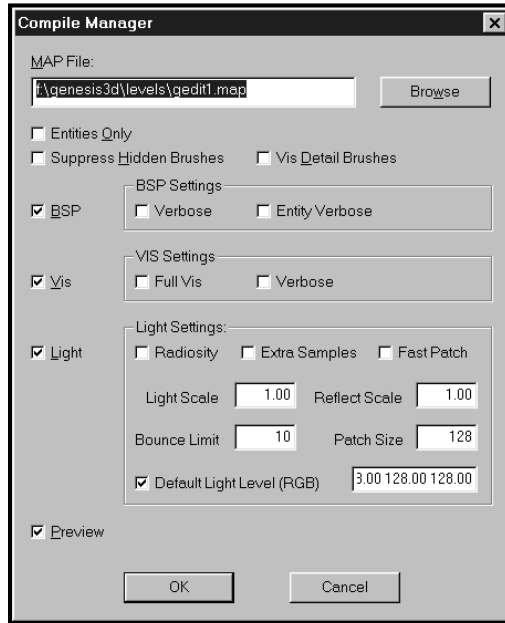
حسناً. الآن وبعد أن قمنا بتعديل حجم الصندوق نريد أن نقوم بوضعه في العالم الثلاثي الأبعاد الذي نحن بصدد إنشائه. نقوم بذلك عن طريق الضغط على مفتاح الإدخال "Enter" ونحن في حالة التعديل (إذا لازلت في حالة الكاميرا غيرها إلى حالة التعديل). بعدها ستلاحظ أن هناك شيء جديداً في الشاشة الرئيسية, وهو أنك سوف تلاحظ بأن الصندوق الذي كان عبارة عن خطوط أصبحت تغطية طبقة لها شكل معين وتستطيع أن تراها في المربع أو النافذة العليا من اليسار Textured.

في هذه النقطة نحن فعلا قمنا بإنشاء غرفة كاملة ولكن بقيت هناك بعض الخطوات الإضافية التي يجب القيام بها قبل أن نستخدم هذا العالم الذي أنشأناه (والذي طبعا لا يحتوي إلا على صندوق واحد يمثل غرفة واحدة لا غير).

- الخطوة الأولى الإضافية هي انك لو نظرت إلى الغرفة التي أنشأناها تحتوي على شكل واحد فقط (أي شكل أو لون خارجي للجدران) وبالتالي فإنها غير واقعية ويصعب معرفة أعلاها من أسفلها. لذلك سنقوم بالتحويل إلى الحالة الثالثة للبرنامج وهي حالة الاختيار عن طريق الضغط على الزر .
- الخطوة الثانية الإضافية هي الضغط على المفتاح Page Down من لوحة المفاتيح أو اختيار الأمر Toggle Brush/Face Mode أحد أوامر Tools من القائمة العلوية. السبب أننا نحتاج أن نختار كل وجه من اوجه هذا الصندوق حتى نستطيع تغيير شكله الخارجي.
- الخطوة الثالثة هي أن نختار أي وجه نشاء عن طريق الضغط عليه من داخل النافذة العليا من اليسار Textured. بعدها نختار قائمة Texture من القوائم على يمين الشاشة ثم نختار الشكل الذي يعجبنا (كلما اخترنا شكلاً معيناً سوف يظهر في اسفل القائمة) ونضغط على الزر Apply. لقد تغير شكل الوجه أو الجدار للشكل الذي اخترته.
- الخطوة الإضافية الأخيرة هي إضافة Entity ونحن في حالة التعديل. سوف نحتاج لاختيار قائمة Template (نفس القائمة التي تحتوي على الفرش) من على القوائم على يمين الشاشة ثم نختار من القائمة DeathMatchStart بجانب المصباح الأصفر. ثم نضغط على المصباح الأصفر , سوف تظهر علامة X زرقاء اللون لتدلنا على مكانها في العالم. وهي تدل على مكاننا في هذا العالم عند

بداية البرنامج. ونستطيع تحريك العلامة عن طريق الضغط عليها بإشارة الفأرة. ثم لا ننسى أن نضغط على الزر **Enter** حتى ندخلها لعالمنا.

إذا قمنا بكل الخطوات السابقة بالشكل الصحيح فإن عالمنا الثلاثي الأبعاد جاهز للإنتاج والتجربه ونقوم بذلك بكل بساطة عن طريق الضغط على الزر  سترى الشاشة التالية :



ثم من النافذة السابقة نضغط على الزر "OK". وسترى الرسالة التالية:

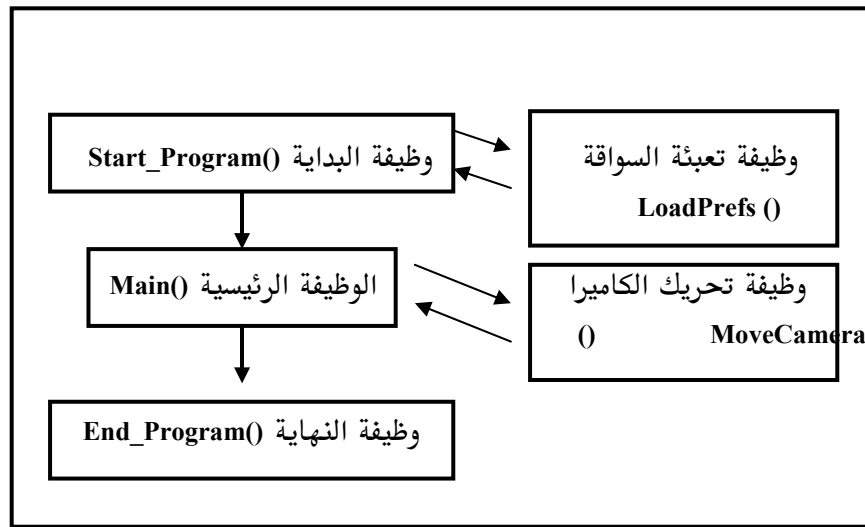


وباختصار تسألك إذا كنت تريد أن تجرب عالمك الثلاثي وكل ما عليك فعله الآن هو الإجابة بنعم عن طريق الضغط على الزر "Yes".

إذا لسبب معين لم تنجح العملية فأعد الخطوات مرة أخرى وكما رأيت هي ليست بالكثيرة ولكنها الخطوات الأساسية لإنشاء أي عالم ثلاثي الأبعاد باستخدام هذه المكتبة.

مثال لاستخدام المكتبة Genesis 3D

بعدما أنشأنا عالمنا نحن الآن مستعدين للدخول إلى الجزء البرمجي لهذه المكتبة وسوف نرى أننا بمجرد استخدام الأوامر المتوفرة لنا من خلال هذه المكتبة (كما كان الحال مع المكتبة AGDX) نقوم بإنشاء برنامجنا. في هذا المثال سوف نتعرف على الأوامر الرئيسية لهذه المكتبة وسوف نكتب برنامج يسمح لنا بالحركة في العالم الثلاثي للأمام وللخلف. كما نقوم بإنشاء كاميرا تكون هي عين المشاهد. إذا كنت متابع الأمثلة السابقة لمكتبة AGDX فإن هذا المثال يتبع نفس المنوال ، فقط سنغير المكتبة إلى Genesis 3D. أما إذا كنت لم تتطلع بعد على أمثلة المكتبة السابقة فإننا في هذا المثال سنقوم باستخدام برنامج Win32 والذي سبق وشرحناه ، مع المكتبة Genesis 3D بنفس طريقة المثال الثالث من أمثلة المكتبة السابقة. أي أننا سوف نضع أوامر Win32 في ملف بشكل منفصل ويكون شكل البرنامج كالتالي :



ولأننا نقوم بالبرمجة تحت أنظمة النوافذ فإن معرفة برمجة Win32 مهمة (هناك فصل خاص (الفصل الثالث) مشروح لها ويستحسن مراجعته).
بعد أن نضع برنامج Win32 في ملف مختلف سنقوم أولاً بإنشاء خمس وظائف كما يلي:

1- الوظيفة الأولى "Start_Program ()" وهي المسؤولة عن كل ما هو مطلوب القيام به عند بداية البرنامج.

2- الوظيفة الثانية "Main ()" وهي مثل وظيفة "Main ()" تحت النظام DOS وفيها سنقوم بكتابة البرنامج الرئيسي.

3- الوظيفة الثالثة "Finish_Program ()" وهي المسؤولة عن كل ما هو مطلوب القيام به عند نهاية البرنامج.

4- الوظيفة الرابعة "LoadPrefs ()" وهي المسؤولة عن قراءة الملف prefs.ini واختيار السواقة الموجودة. هذا الملف هو عبارة عن ملف من نوع txt سنقوم بكتابته ويحتوي على التالي :

G
640
480

الحرف الأول يدل على السواقة التي نود استخدامها سواء كان "G" لمسرعات 3dfx أو "S" لتكنولوجيا DirectX أو "S" Software والسواقة (أو الأداة) هي التي تحدد سرعة البرنامج.

بعد ذلك عرض الشاشة المطلوب ثم ارتفاعها.

5- الوظيفة الخامسة "MoveCamera ()" وهي المسؤولة عن حركة الكاميرا.

طبعا سنقوم بإنشاء ملفين إضافيين أحدهما يحمل تعاريف البرنامج المطلوب استخدمها (سنطلق عليه الاسم "main.h") والثاني نضع فيه هذه الوظائف (سنطلق عليه الاسم "Main.cpp") وسوف نلقي نظرة إلى هذه العملية لاحقا.

بعض التعديلات على الملف win32.CPP :

الآن سوف نحتاج إلى القيام بالتعديلات المناسبة في البرنامج الرئيسي والذي يحتوي على برنامج win32 كما يلي :

- أولا نبدأ ببرنامج المثال الأول ونقوم بنقل ما يلي إلى ملف التعاريف :

```
#include <windows.h>
#include <windowsx.h>
```

ونضع بدلها أمر النظر إلى ملف التعاريف :

```
#define Ex1_CPP
#include "main.h"
```

- في الخطوة "1" نقوم بنقل التعريف "HWND hWnd;" وهو المتغير الذي يحمل عنوان برنامجنا إلى ملف التعاريف.
- في الخطوة "4" سنقوم بتغيير أحد مميزات نافذة البرنامج من "WS_OVERLAPPEDWINDOW" إلى "WS_POPUP" لأننا نستخدم الشاشة بكاملها (نضعه كما هو في حالة تغيير البرنامج لاستخدام نافذة من

نوافذ (Windows) هذه الخطوة ليست ضرورية ولكنها مستحبة. لو انك جربت المثال الأول سوف ترى شاشة بيضاء "لجزء من الثانية" ثم تختفي ، ولذلك نحن نقوم بهذه الخطوة حتى لا نرى تلك الشاشة المزعجة مطلقا في البرامج التي تستغل الشاشة بكاملها. (تحذير: يجب إرجاع هذه الميزة إلى حالتها الأولى في حالة أن البرنامج يستخدم نافذة ، وإلا فإنك لن ترى شيئا على الشاشة)

- في الخطوة "6" نقوم بنقل جميع خطوات إنشاء كائن الشاشة إلى الوظيفة "Start_Program ()" ونضع بدلها أمر الذهاب إليها.
- في الخطوة "7" سنقوم ببعض التعديلات حتى نستطيع استخدام وظيفتنا الجديدة "Main ()" كما يلي:

```
while(1)
{
    if(PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))
    {
        if(!GetMessage(&msg, NULL, 0, 0 )) return msg.wParam;
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    else if(bActive)
    {
        Main();
    }
    else WaitMessage();
}
```

باختصار يقوم الكمبيوتر بذهاب إلى الوظيفة Main() كلما كان المتغير "bActive" يساوي صحيح "TRUE".

- وأخيرا في الخطو "10" نقوم بنقل خطوات رسالة إنهاء البرنامج "WM_DESTROY" إلى الوظيفة "Finish_Program ()" ونضع بدلها أمر الذهاب إليها.

ملف التعاريف Main.H :

في هذا الملف نقوم بوضع تعاريف البرنامج ، هذا يجعل برامجنا أكثر سهولة للفهم وأقل تعقيدا ونرى الملف كما يلي :

```
#ifndef MAIN_H
#define MAIN_H

#ifdef WIN32_CPP
#define DECLARE
#else
#define DECLARE extern
#endif

#include <windows.h>
#include <windowsx.h>
#include <stdio.h>
#include <genesis.h>

void Start_Program(void);
void Finish_Program(void);
void Main(void);
void MoveCamera(void);
void LoadPrefs(void);

DECLARE HWND hWnd;
DECLARE BOOL bActive;
DECLARE int spin; // = 0; //whether or not we are spinning
DECLARE FILE *stream;

LRESULT CALLBACK WndProc(HWND hWnd, UINT iMessage,
WPARAM wParam, LPARAM lParam);
```

```

DECLARE geEngine *Engine;
DECLARE geDriver_System *DrvSys;
DECLARE geDriver *Driver;
DECLARE geDriver_Mode *Mode;
DECLARE geWorld *World;
DECLARE geCamera *Camera;
DECLARE geRect Rect;
DECLARE const char *drvname;
DECLARE MSG Msg;
DECLARE geXForm3d Xform;
DECLARE geXForm3d ViewXForm;
DECLARE long Width, Height;

DECLARE char TempName[1024];
DECLARE geVFile *MainFS, *map;

//Our Clients Width And Height, and driver(loaded from prefs.ini)
DECLARE int CWidth; //=640;
DECLARE int CHeight; //=480;
DECLARE char ourdriver; //= 'G';

#endif

```

وفية نقلنا تعاريف البرنامج (هنا أيضا سنقوم بإضافة أي تعاريف جديدة في المستقبل). الأوامر "#define" و "#ifndef" و "#ifdef" و "#else" و "#endif" وضعت هنا بقصد التأكد أن الكمبيوتر لن يقوم بتعريف المتغيرات أكثر من مرة وبالتالي لا تحدث أية مشاكل.

أما بقية التعاريف فكما نرى أنها تبدأ بالعبارة **DECLARE** ونحن سبق واخبرنا المصرف (Compiler) في أعلى الملف أن هذه العبارة تعني الأمر **extern** ونحتاج لهذا الأمر قبل أي تعريف للكائنات أو المتغيرات المستخدمة في الملف (أيضا كما كان الحال مع المكتبة AGDX) حتى نستطيع استخدامها في أي نقطة من البرنامج بشكل عام وهذه أحد قواعد لغة السي المحسنة. بعد ذلك نرى أننا استخدمنا

أسماء الكائنات التي نحتاجها , وبكل سهولة نرى الكائن الذي نحن بحاجة له ونقوم بإنشائه في ملف التعاريف. ولكي تأخذ فكرة واضحة سوف نقوم أولاً في هذا المثال بالنظر إلى بعض كائنات هذه المكتبة التي قمنا باستخدامها. طبعاً الطريقة المثلى هي الرجوع إلى مرجع هذه المكتبة كلما احتجت لوظيفة معينة في برنامجك. في هذا المثال نحن استخدمنا من الكائنات ما يلي :

- **FILE** الكائن المسؤول عن قراءة وتخزين الملفات من وعلى القرص الصلب هذا الكائن جزء من لغة السي وليس جزء من هذه المكتبة.
- **geEngine** الكائن المسؤول عن الإخراج للبرنامج ويحتوي على الكائنات ثنائية الأبعاد **geDriver_System** و **geWorld** و **geBitmap** والتي عن طريقها نستطيع أن نتحكم في كل ما هو ثنائي الأبعاد مثل الصور والكتابة الخ ونستطيع أن نعتبره مشابه في وظيفته للعنصر **DirectDraw** من عناصر **DirectX**
- **geDriver_System** هذا الكائن هو أحد أعضاء الكائن **geEngine** وعن طريقة نستطيع أن نتعرف على السواقات (أو الأدوات سواء كانت **Software** , **DirectX** أو **3DFX**) المتوفرة على الجهاز وسوف نستخدمه مع الكائن التالي للتأكد من وجود السواقة المطلوبة على الجهاز.
- **geDriver** عن طريق هذا الكائن نختار نوع السواقة التي سوف نستخدمها. وفي هذا المثال سوف نقوم بكتابة ملف نطلق عليه **prefs.ini** عن طريق برنامج كتابة مثل **Notepad** أو أي

برنامج كتابه آخر ويحتوي على السواقة المطلوبة (راجع عمل الوظيفة الرابعة في هذا المثال). بعد أن نختار السواقة المناسبة نستخدم الكائن السابق للتأكد أنها موجودة على الجهاز. مثلا افرض أننا اخترنا سواقة سرعات 3dfx ثم أن الكائن السابق لم يخبرنا بوجودها على الجهاز وبذلك نستطيع أن نخرج من البرنامج بدون حدوث أي مشاكل أو اختيار سواقة أخرى تكون متوفرة.

- **geDriver_Mode** بعد أن اخترنا السواقة المناسبة وتأكدنا أنها موجودة على الجهاز. نستخدم الكائن السابق مع هذا الكائن للتأكد أن الصفاء المطلوب للشاشة والذي سبق وكتبناه في الملف prefs.ini متوفر مع السواقة التي اخترناها سواء كان 320x240 أو 640x480.... الخ
- **geWorld** حسناً الآن نحن على استعداد للنظر في تعبئة عالمنا الثلاثي وهذه هي وظيفة هذا الكائن الذي يأخذ الملف " bsp .اسم الملف" والذي أنشأناه في المصمم.
- **GeRect** عن طريق هذا الكائن سنحدد المربع أو المستطيل على الشاشة الذي ستستخدمه الكاميرا للعرض.
- **geCamera** هذا كائن الكاميرا والذي سوف يستخدم الكائن السابق لعرض العالم الثلاثي الموجود فيه مع اعتبار موقع الكاميرا منه.
- **geXForm3d** هذا الكائن هو عبارة عن مصفوفة 4x4 نقوم عن طريقها بتحديد موقع الكاميرا من العالم.

- `geVec3d` هذا الكائن شبيهه بالكائن السابق وهو عبارة عن متجهات (`3d` vector) ثلاثية الأبعاد نقوم عن طريقها بتحديد موقع أي نقطة في العالم ويقوم هذا الكائن بمساعدتنا للقيام بذلك.
- `geVFile` هذا الكائن شبيهه بالكائن الأول (`FILE`) ولكنه جزء من هذه المكتبة وهو المهتم بالتعامل مع الملفات المطلوب استخدامها ويقوم بإضافة مميزات لا توجد في الكائن الأول وضرورية لنا في برمجة هذه المكتبة.

الملف الرئيسي Main.cpp :

بعد أن تعرفنا على الكائنات التي سوف نستخدمها في مثالنا هذا , تعالوا ننظر إلى الملف الرئيسي ونرى كيف استخدمنا تلك الكائنات. نحن طبعا قمنا بإنشاء ممثلين لتلك الكائنات في ملف التعاريف وطريقة إنشاء الممثلين للكائنات شبيهة جدا بطريقة إنشاء المتغيرات فكل ما نقوم به هو إعطاء اسم الكائن ثم اسم الممثل كما هو الحال مع المتغيرات فنحن نعطي نوع المتغير ثم اسمه كما يلي :

- في حالة المتغيرات مثلا نقوم بكتابة ما يلي في برنامجنا : (`Int value;`)
المتغير اسمه `value` من نوع `int`
- في حالة الممثلين للكائنات نقوم بكتابة ما يلي في برنامجنا :
(`geEngine *Engine;`) الممثل اسمه `Engine` من نوع الكائن `geEngine` ولاحظ النجمة قبل اسم الممثل والتي تدل أن هذا الكائن سيُحفظ في الذاكرة وسوف يكون مؤشرا للكائن.

الآن سوف تعلم عزيزي القارئ لماذا قمنا بكل ذلك ، لأن برنامجنا الرئيسي سوف يصبح اكثر سهوله. بل أنك إذا كنت تعرف البرمجة على النظام Dos سوف تكتشف مدى تقاربه مع هذا البرنامج. لاحظ كذلك أن ملف برنامج win32 لن يتغير بشكل رئيسي من الآن فصاعد ، ولذلك لن نهتم به كثيرا كما أن التغيير سوف يكون هنا في برنامجنا الرئيسي فقط.

طبعا الملف الرئيسي سوف يحتوي على الوظائف الخمس والتي سبق وأشرنا إليها ونبدأ بشرح الملف من الأعلى إلى الأسفل مروراً بالوظائف الخمس التي ذكرناها ونشاهد الملف الرئيسي كما يلي:

```
#include "main.h"

void Start_Program(void)
{
    bActive=TRUE;

    LoadPrefs();

    Engine = geEngine_Create(hWnd, "RPG", ".");

    DrvSys = geEngine_GetDriverSystem(Engine);
    if (!DrvSys) MessageBox(NULL,"No DrvSys", "Error",MB_OK);
}
```

كما لاحظت أننا بدأنا أولاً بأمر المصرف بالنظر إلى ملف التعاريف في السطر الأول ثم بعد ذلك بدأنا بالوظيفة الأولى "وظيفة البداية".

في وظيفة البداية جعلنا المتغير bActive يساوي القيمة TRUE "صحيح" وسيظل هذا المتغير يحمل هذه القيمة مادام البرنامج يعمل ونستخدم هذا المتغير في حالة إذا أردنا التعرف على حالة البرنامج. والسبب يعود إلى أنه في نظام النوافذ قد يكون

هناك أكثر من برنامج يعمل في نفس الوقت ووجود متغير مفيد لنا. نحن لن نستخدمه في هذا المثال ولكن وجوده مهم إذا أردنا تطوير المثال أكثر.

بعد ذلك مباشرة امرنا المصرف بالذهاب إلى الوظيفة الخامسة في هذا البرنامج " LoadPrefs(); حتى نقرأ محتويات الملف prefs.ini (سننظر لهذه الوظيفة في نهاية شرح هذا البرنامج) .

بعد ذلك قمنا بإنشاء الكائن الأول geEngine وعن طريقة قمنا بإنشاء الكائن الثاني DrvSys لاحظ الترتيب حيث أننا أيضاً سنستخدم الكائن الثاني لإنشاء الكائن الثالث وهكذا ترى علاقة الكائنات ببعضها البعض ولماذا نحتاج لإنشائها بهذا الترتيب المعين.

```

Driver = geDriver_SystemGetNextDriver(DrvSys, NULL);
if (!Driver) MessageBox(NULL,"No Driver","Error",MB_OK);

while(1) {
geDriver_GetName(Driver, &drvname);

if (drvname[0] == ourdriver) break;

Driver = geDriver_SystemGetNextDriver(DrvSys, Driver);
if (!Driver) _exit(-1);
}

```

في هذا الجزء قمنا بإنشاء كائن الأداة أو السواعة (Driver) وكما ترى أننا استخدمنا الكائن DrvSys الذي أنشأناه قبل قليل.

بعد ذلك وضعنا الأمر الشرطي if لتتأكد أن هناك سواعة (أو سواقات) بالجهاز المستخدم فإذا كان كذلك فننتقل إلى الخطوة الثانية وإلا نخرج من البرنامج برسالة خطأ.

في الخطوة الثانية وبما أن هناك احتمال وجود أكثر من سواقة بالجهاز فإننا نستخدم الأمر الشرطي `while(1)` ويعني قم بتنفيذ الأوامر التالية مادام الشرط صحيح مما يضع البرنامج في دورة (هذا الأمر شبيه بالأمر السابق `if` “ ولكنه يختلف في أن البرنامج يظل في دورة مادام الشرط صحيح).

في الأمر الذي يليه `geDriver_GetName(Driver, &drvname);` قمنا بالحصول على أحد السواقات المتوفرة على الجهاز ووضعناها في المتغير `drvname` ثم في الأمر الذي يلي ذلك قمنا باستخدام الأمر الشرطي `if` “ لتتأكد أن السواقة التي اخترناها في الملف `prefs.ini` هي نفس السواقة الموجودة في المتغير `drvname` فإذا كان ذلك صحيحاً فقد وجدنا سواقتنا المطلوبة ونخرج من هذه الدورة للأمر الشرطي ، أما إذا كان غير صحيح نذهب إلى الأمر الذي يلي ذلك :

```
Driver = geDriver_SystemGetNextDriver(DrvSys, Driver);
```

وهو نفس الأمر الذي شرحناه في بداية هذا الجزء ويقوم بالتأكد انه لازال هناك سواقات أخرى إضافية يجب أن نتأكد منها ونضعها في الكائن `Driver`. وتعود العملية من جديد.

```
Mode = geDriver_GetNextMode(Driver, NULL);

while(1) {
if (!Mode) MessageBox(NULL,"No Mode","Error",MB_OK);

geDriver_ModeGetWidthHeight(Mode, &Width, &Height);
if (Width == CWidth && Height == CHeight) break;

Mode = geDriver_GetNextMode(Driver, Mode);
}
```


في هذا الجزء (لا تنسى أننا لازلنا في وظيفة الإعداد) سنقوم بعملية مشابهة للجزء السابق ولكن ليس مع السواقات الموجودة في الجهاز ولكن مع صفاء الشاشة الذي تستطيع السواقة أن تعطيه على الجهاز ومقارنته بالصفاء الموجود في الملف `prefs.ini` (سواء كان `320x240` أو `640x480`... الخ).

```
if (!geEngine_SetDriverAndMode(Engine, Driver, Mode))
{
    MessageBox(NULL, "Set Driver/Mode failed", "Error", MB_OK);
    _exit(-1);
}
```

إذا كان كل شيء على ما يرام فإننا الآن نملك سواقة سبق واخترناها في الملف `prefs.ini` وكذلك صفاء معين للشاشة نحن اخترناه أيضا. في هذا الجزء سنستخدم الأمر `geEngine_SetDriverAndMode` ليقوم بالتأكد أن المكتبة قادرة على استخدام السواقة بالصفاء المطلوب وأن كل شيء على ما يرام وإلا فإننا نخرج من البرنامج برسالة خطأ سببها عدم توافق صفاء الشاشة المطلوب مع السواقة.

```
GetCurrentDirectory(sizeof(TempName), TempName);

MainFS = geVFile_OpenNewSystem(NULL, GE_VFILE_TYPE_DOS,
    TempName,
    NULL, GE_VFILE_OPEN_READONLY |
    GE_VFILE_OPEN_DIRECTORY);

map = geVFile_Open(MainFS, "levels\\genvs.bsp",
    GE_VFILE_OPEN_READONLY);

World = geWorld_Create(map);
```

```
if (!World)
{
    MessageBox(NULL,"No World","Error",MB_OK);
    _exit(-1);
}
if (!geEngine_AddWorld(Engine, World))

{
    MessageBox(NULL,"Failed to add World","Error",MB_OK);
    _exit(-1);
}
```

في هذا الجزء سنهتم بالعالم الثلاثي الأبعاد والذي نود استخدامه في مثالنا. كما سنقوم بتعبئة الملف الذي سبق وأنشأناه في برنامج المصمم للعالم الثلاثي الأبعاد. طبعاً نقوم بالخروج من البرنامج إذا لم نجد الملف أو كان هناك خطأ. ثم نستخدم الممثل World لتعبئة العالم الثلاثي (مع خريطة تابعة له) بعد ذلك نضيف هذا العالم للكائن الرئيسي geEngine.

```
Rect.Left = 0;
Rect.Right = CWidth-1;
Rect.Top = 0;
Rect.Bottom = CHeight-1;

Camera = geCamera_Create(2.0f, &Rect);
if (!Camera)
{
    MessageBox(NULL,"No Camera","Error",MB_OK);
    _exit(-1);
}

}
```

و أخيرا وصلنا للجزء الأخير من وظيفة البداية , في هذا الجزء سنهتم بإنشاء الكاميرا وإعطائها مكانها الأولي. وكما ترى سنستخدم المصفوفة التي تحدثنا عنها لوصف المكان بالتحديد.

في البداية استخدمنا الكائن Rect لتحديد المربع أو المستطيل الذي سيكون واجهة الكاميرا.

بعد ذلك قمنا بإنشاء كائن الكاميرا باستخدام ذلك المستطيل. وهنا أيضا نقوم بالخروج من البرنامج عند حدوث أية أخطاء.

```
void Finish_Program(void)
{
    //shut it all down and quit
    geEngine_Activate(Engine, GE_FALSE);

    geCamera_Destroy(&Camera);

    geEngine_RemoveWorld(Engine, World);

    geWorld_Free(World);

    geEngine_ShutdownDriver(Engine); //Works again in beta 4

    geEngine_Free(Engine);

    PostQuitMessage (0);
}
```

أما وظيفة النهاية فنستخدمها عند نهاية البرنامج (أو عند الضغط على المفتاح Esc) ونقوم من خلالها بالتأكد من تحرير الذاكرة عن طريق إزالة الكائنات والقيام بكل ما هو ضروري لذلك.

```

void Main(void)
{
    MoveCamera();

    if (GE_FALSE == geEngine_BeginFrame(Engine, Camera
    ,GE_TRUE)) Finish_Program();

    if (GE_FALSE == geEngine_RenderWorld(Engine, World, Camera,
    0.0f)) Finish_Program();

    if (GE_FALSE == geEngine_EndFrame(Engine)) Finish_Program();
}

```

وهذه الوظيفة الرئيسية كما نلاحظ أنها لا تحتوي على الكثير من الأوامر لأن أغلب العمل كان في إعداد المكتبة والاهتمام بالوظائف الفرعية كوظيفة تحريك الكاميرا. ولكن تختلف الوظيفة الرئيسية في عملها قليلا عن الوظائف الأخرى في أنها تعمل بشكل متكرر , بمعنى أنها تبدأ من أول أمر وتنتهي بأخر أمر فيها ثم تعود مرة أخرى إلى أول أمر و هلم جرا.

ولو نظرنا إلى أول أمر فيها فنرى انه أمر استدعاء وظيفة الكاميرا (سننظر لهذه الوظيفة بعد قليل) وهو الأمر MoveCamera() وفيه نترك وظيفة الكاميرا تقوم بالتغيرات المناسبة للكاميرا في المشهد –ويتم ذلك بشكل متكرر طبعا–

في الأمر الذي يلي ذلك geEngine_BeginFrame(Engine, Camera ,GE_TRUE) نقوم بالتأكد بأن الكائن الرئيسي geEngine مستعد للإطار (Frame) الجديد للمشهد, تذكر أننا نقوم بتكرار هذه الوظيفة وبالتالي فكل الأوامر التي بها ستكون مكررة كذلك. ولأننا نريد الحركة أن تكون سلسلة بقدر المستطاع فإننا نريد تقريبا 30 إطار بالثانية (أو ثلاثين صورة بالثانية فالإطار هو كل صورة نراها على الشاشة)

أو أكثر ويعتمد ذلك على السواقة المستخدمة وهذا الأمر يقوم بالتأكد أن الكائن الرئيسي مستعد لكل إطار نريد عرضه وإلا فإننا نخرج من البرنامج.

في الأمر الذي يلي ذلك `geEngine_RenderWorld(Engine, World, Camera, 0.0f)` فيجب أن يتبع الأمر السابق وما يقوم به هو وضع كل ما هو مطلوب عرضة على الشاشة إلى السطح الخلفي (`back-buffer`) وذلك حتى يهيئنا للأمر الأخير في هذا الوظيفة. طبعاً أن حصل أي خطأ فإننا نخرج من البرنامج.

في الأمر الأخير لهذه الوظيفة `geEngine_EndFrame(Engine)` فإننا نقوم بقلب محتويات السطح الخلفي للسطح الرئيسي (شبيه بوظيفة `flip()` في مكتبة AGDX) عندها نستطيع أن نرى الصورة على الشاشة.

وهكذا يتكرر تنفيذ هذه الأوامر حتى ننهي البرنامج.

```
void MoveCamera(void)
{
    POINT pos;
    geVec3d Pos;

    GetCursorPos(&pos);
    ScreenToClient(hWnd, &pos);
}
```

الآن نبدأ في شرح وظيفة حركة الكاميرا وكما نرى أننا نبدأ أولاً بإنشاء بعض المتغيرات. المتغير `pos` هو عبارة عن نقطة على الشاشة سنستخدمها للتعبير عن موقع إشارة الفأرة والمتغير `Pos` هو عبارة عن ممثل للكائن `geVec3d` سوف نستخدمه للتعبير عن موقع الكاميرا.

نبدأ أولاً في استخدام المتغير الأول `pos` للحصول على موقع إشارة الفأرة والتي على أساسها سوف نبني حركتنا في العالم ثلاثي الأبعاد (إذا كانت الإشارة في أعلى الشاشة فإننا نتحرك إلى الأمام والعكس صحيح).

```

if (pos.y > ((CHeight/2) + 50)) { //is it to the top?
Xform.Translation.Z += 3.0f; //move forward
}
if (pos.y < ((CHeight/2) - 50)) { //is it to the bottom?
Xform.Translation.Z -= 3.0f; //move backward
}

Pos = Xform.Translation;

// Clear the matrix
geXForm3d_SetIdentity(&ViewXForm);

geXForm3d_Translate(&ViewXForm, Pos.X, Pos.Y, Pos.Z);

geCamera_SetWorldSpaceXForm(Camera, &ViewXForm);
}

```

في الجزء الثاني والأخير من هذه الوظيفة وبعدما حصلنا على موقع إشارة الفأرة على الشاشة فإننا نقوم بفحص الموقع عن طريق الأمر الشرطي `if` على المحور الصادي "Y" لأنه يدل على أعلى الشاشة وأسفلها وبناء على الموقع فإننا نحرك الكاميرا أما للأمام أو للخلف.

بعد أن عرفنا موقع الكاميرا الجديد الموجود الآن في المتغير `Xform` فإننا نقوم بنقل محتوياته للمتغير `Pos` والذي أنشأناه في بداية هذه الوظيفة.

بعد ذلك نستخدم الأوامر الثلاث الأخيرة لتحريك موقع الكاميرا على الإحداثيات
.X , Y , Z

```
void LoadPrefs(void)
{
    stream = fopen("prefs.ini", "r");//open as read only
    fscanf(stream, "%s", &ourdriver);//get our information
    fscanf(stream, "%d", &CWidth);
    fscanf(stream, "%d", &CHeight);
    fclose(stream); //dont forget to close
}
```

و أخيرا الوظيفة الأخيرة LoadPrefs() وعن طريقها نقوم بقراءة الملف الذي سبق
وكتبناه (باستخدام أي برنامج كتابة) وهو الملف prefs.ini وفيه كتبنا حرف معين
(اخترناه من مجموعة أحرف تدل على السواقة وسبق وشرحنا هذا في بداية المثال)
ثم بعد ذلك العرض الذي نريده للشاشة ثم الارتفاع.

هذه هي كل محتويات الملف الرئيسي أو بالأحرى هذا هو كل برنامج المثال الثالث.
ألا تلاحظ السهولة بعد أن تخلصنا من برمجة win32 (نحن طبعا لم نتخلص منها
ولكننا وضعناها في ملف خاص بها ولن نهتم إلا بإلحاق ذلك الملف في برامجنا
القادمة). وأحب أن أذكر مرة أخرى عزيزي القاري أن الشفرة البرمجية لهذا
المثال موجودة على القرص المدمج الملحق مع الكتاب فلا داعي لإعادة طباعته مرة
أخرى.

وأخير بعد إنتاج ملف التشغيل "exe" لبرنامجنا يجب أن نتأكد أن الملفات :

softdrv2.dll ❖
 SoftDrv.dll ❖
 GlideDrv.dll ❖
 Genesis.dll ❖
 GBSPLib.dll ❖
 D3DDrv.dll ❖
 GBSPLib.dll ❖

موجودة في نفس المكان الذي يوجد به ملف التشغيل أو أنها من ضمن ملفات

System تحت نظام Windows.

ثم يجب أن لا ننسى الملف :

❖ prefs.ini (هذا ملفنا الذي كتبناه باستخدام أي برنامج كتابة)

انظر للمثال الملحق لتعرف ما أعنيه.

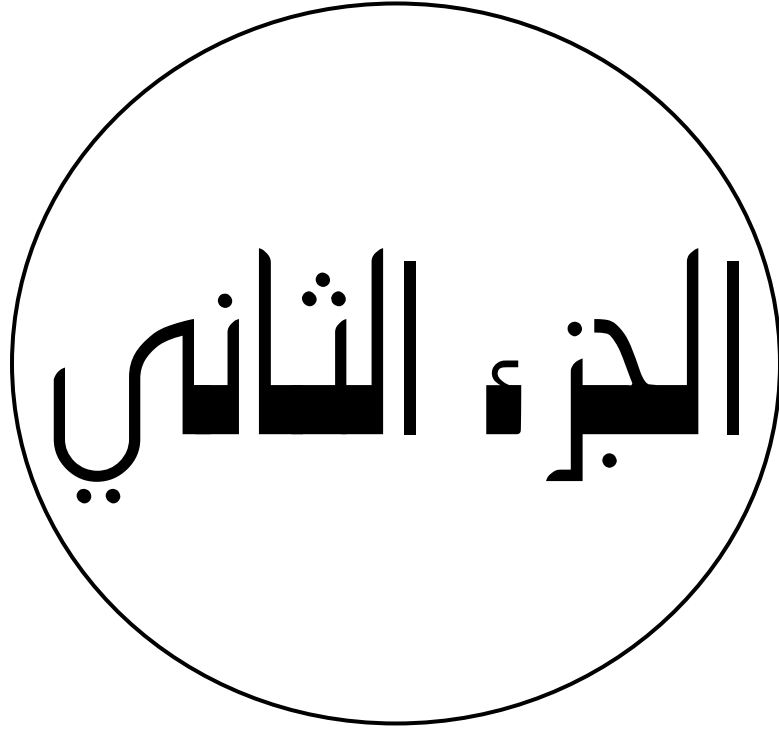
في الحقيقة نحن في هذا المثال لم نحقق إلا حركة بسيطة للأمام وللخلف ولكنها بداية ممتازة لإنشاء أي برنامج لهذه المكتبة. لأن اصعب خطوة هي في كيفية بداية استخدامها. وأنا متأكد انك كلما تعمقت في هذه المكتبة كلما وجدتتها مكتبة عملاقة تستحق أن تنافس المكتبات باهظة الثمن. ولكنها في نفس الوقت سهلة الاستعمال. وربما تعجبت من أننا لم نتعمق كثيراً في هذا الكتاب لشرح هذه المكتبة وذلك لأن موضوع هذا الكتاب لا يؤهلنا للكتابة عن مكتبة واحدة فقط. ولأن هذه المكتبة تحتاج لكتاب كامل لوحدها. ولكنني أصررت على أوراها في هذا الكتاب لقيمتها للمبرمج ولصلتها العميقة بموضوع هذا الكتاب. في الجزء الثاني من هذا الكتاب

سنتكلم عن العنصر Direct3D والذي هو أحد الأجزاء المهمة من هذه المكتبة ويمكنك الإطلاع على شفرتها البرمجية إذا أردت (ستجد الشفرة البرمجية في نفس الملف الذي ركبت فيه هذه المكتبة على القرص الصلب في جهازك). كما ستجد الشفرة البرمجية للمثال الموجود على القرص في نفس المكان.

وحتى تبدأ استعمال هذه المكتبة حالياً ، هناك نسخة "كاملة" من برنامج الأبعاد الثلاثية TrueSpace الإصدار الأول في الملف Extra على القرص المدمج. وكذلك مع هذا البرنامج ستجد برنامج صغير يسمى "trueGene.exe" مع الشفرة البرمجية التابعة له كذلك والذي تستطيع من خلاله تحويل المجسمات ثلاثية الأبعاد للبرنامج TrueSpace لمثليين تستطيع المكتبة Genesis 3D استخدامها. وبما أن البرنامج TrueSpace يستطيع قراءة العديد من الملفات ثلاثية الأبعاد المختلفة ، بما في ذلك ملفات البرنامج 3D Studio فإنك ستكون على مقدره للتعامل مع الكثير من المجسمات ثلاثية الأبعاد للبرامج الثلاثية المتعددة.

هذا لا يعني نهاية المطاف لهذه المكتبة ولكن بدايته ، إذا كان لديك عزيزي القارئ أي استفسارات عن هذه المكتبة أو أي موضوع في الكتاب فلا تتردد في مراسلتي على عنواني الإلكتروني أو أن تلقي نظر لصفحتي على الإنترنت :

<http://www.Arabgames.com>



الفصل الحادي عشر

القوائم المتصلة

Linked Lists

القوائم المتصلة – Linked Lists



الحجز المتغير للذاكرة :

عند استخدام الحركة على الكمبيوتر فإننا نقوم برسم الشكل المراد تحريكه أكثر من مرة حتى تنتج الحركة. فلو أردنا تحريك سفينة فضاء مثلاً فإننا نحتاج إلى تخزين الصور المختلفة من أشكال تلك السفينة في ذاكرة الكمبيوتر ، وبمعرفتنا حجم كل صورة فإننا نستطيع تحديد الحجم الكلي للذاكرة المطلوبة. وعليه نستطيع أن نستخدم متغير من نوع "array" لتخزين تلك الصور وحجز الذاكرة المطلوبة لها. ولكن لا تجدي هذه الطريقة عند عدم معرفتنا بالحجم المطلوب حجزه في الذاكرة. فلو أردنا أن نستخدم طلقات رصاص في لعبة معينة ولا نعلم كم عدد الطلقات التي سوف نستخدمها لأن ذلك يعتمد على عدد مرات ضغط اللاعب على المسدس خلال عمل البرنامج ، لذلك فإننا لا نستطيع حجز الذاكرة المطلوبة. وعليه سوف نرى فائدة القوائم المتصلة في إعطائنا القدرة على حجز و تحرير الجزء المطلوب من الذاكرة خلال عمل البرنامج (تسمى هذه الطريقة في التعامل مع الذاكرة بالحجز المتغير للذاكرة أو **Dynamically allocated memory**) وذلك لأن القوائم المتصلة تسمح لنا بإضافة ومسح أي عدد من مكوناتها خلال عمل البرنامج (هذه الطريقة المستعملة في الألعاب وغيرها من برامج الصوت والصورة للتحكم في

الذاكرة ، ولكن استخدام القوائم المتصلة لا يقتصر على ذلك وتستطيع بعد استيعاب طريقة عمل هذه القوائم في استخدامها لأشياء أخرى مفيدة في برامجك).
ما هي القوائم المتصلة ؟

القوائم المتصلة هي عبارة عن معلومات مركبة (data structure) تحتوي على مجال لتخزين المعلومات و مؤشر أو مؤشرات (المؤشر "pointer" هو عبارة عن متغير يحمل رقم لعنوان معين في الذاكرة—عنوان المعلومة أو نقطة الاتصال المخزنة في هذه الحالة). وتسمى كل معلومة في القوائم المتصلة بـ "node" أو نقطة اتصال. والمؤشر يُوْشر إلى نقاط الاتصال الأخرى في القائمة. ومن خلال تسلسل نقاط الاتصال عن طريق المؤشرات نستطيع المرور بكل القائمة عن طريق اتباع تلك المؤشرات. ونستطيع أن نلخص الفكرة أن لكل معلومة في القائمة (أو في الذاكرة بما أن القائمة ستكون موجودة بها) مؤشر للمعلومة التي تليها (لماذا نقوم بذلك؟) نقوم بذلك لكي نربط جميع المعلومات ببعضها البعض ومعرفة مكان كل معلومة في القائمة حتى نستطيع إضافتها أو إزالتها عندما نشاء (قد تكون المعلومة عبارة عن كائن أو متغير من نوع struct أو شعاع لصورة أو متغير يحمل قيمة معينة وهكذا).

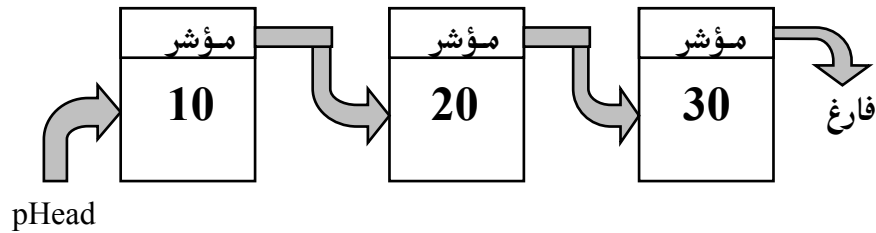
هناك نوعين من القوائم المتصلة : القوائم الأحادية وهي تحتوي على مؤشر واحد لكل نقطة اتصال بحيث يكون مؤشرا إلى نقطة الاتصال التالية في القائمة. و القوائم الثنائية وهي تحتوي على مؤشرين لكل نقطة اتصال ،مؤشر إلى نقطة الاتصال التالية في القائمة ومؤشر إلى نقطة الاتصال السابقة في القائمة. وباستخدام مؤشرين نستطيع المرور بكل القائمة من كلا الاتجاهين.

في البداية سوف نتعرف على النوع الأول (النوع الثاني ينطبق عليه نفس الشرح مع بعض الإضافات التي سوف نذكرها فيما بعد).

كل نقطة اتصال في القوائم المتصلة هي عبارة عن متغير من نوع typedef struct كما يلي:

```
typedef struct NODE
{
    int iValue;
    struct NODE * pNodeNext;
} NODE;
```

مثلا لو كان عندنا ثلاث نقاط اتصال يحتويون على الأرقام 10 ، 20 و 30 كما يلي:



في هذه الحالة :

```
NODE * pHead;
```

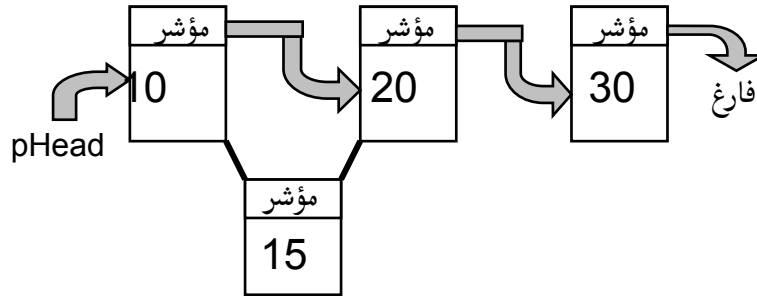
وكما نرى أن المربعات تشكل نقاط الاتصال و الأسهم تشكل المؤشرات بحيث تربط كل مربع (أي كل نقطة اتصال) بالمربع الذي يليه. وهناك نوع من المؤشرات يطلق

علية المؤشر الفارغ (NULL أو فارغ) وهو ما عبرنا عنه بالسهم المتجهة إلى الأسفل بعد الرقم 30 ويشير إلى نهاية القائمة.

لاحظ كذلك أننا استخدمنا متغير أطلقنا عليه pHead (المؤشر الأول) وذلك كي نستطيع مراقبة نقطة الاتصال الأولى في القائمة.

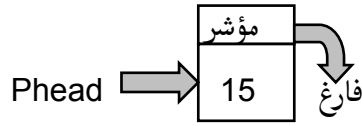
إضافة نقطة اتصال إلى القائمة المتصلة :

تعرفنا على أهمية القائمة المتصلة للتحكم في الذاكرة بحيث تعطينا القدرة على ترتيبها في كل الأوقات وذلك عن طريق إضافة نقاط اتصال كلما أردنا ذلك ولكي نقوم بهذه العملية فإننا نقوم بكتابة وظيفة (function) تستطيع إضافة نقطة الاتصال وتحتاج تلك الوظيفة لمعرفة المكان المطلوب للإضافة وبالتالي تقوم بإعادة ترتيب المؤشر السابق لمكان نقطة الاتصال الجديدة. دعونا ننظر إلى المثال التالي لمعرفة كيفية القيام بهذه العملية :



كيف يمكننا إضافة نقطة الاتصال والتي تحمل الرقم 15 في هذه الحالة ؟

أولا يجب أن نمر بكل القائمة لنقرر مكان نقطة الاتصال الجديدة. ولكي نقوم بذلك دعونا نقوم أولا بتأكد من حالة القائمة ، هل هي فارغة ؟ فإذا كانت كذلك فإن المؤشر pHead تكون له قيمة فارغة (NULL) وذلك لأنه المؤشر الأول وكذلك الأخير في القائمة. وفي هذه الحالة فإن إضافة نقطة اتصال تصبح عملية بسيطة لأن كل ما نقوم به هو وضع المؤشر pHead باتجاه النقطة الجديدة و مؤشر نقطة الاتصال الجديدة له قيمة فارغة كما يلي :



المشكلة في حالتنا هذه أن القائمة ليست فارغة وبالتالي فإن الأمور لن تكون بهذه السهولة وعلينا أن نمر بالقائمة (من جهة واحدة مبدئياً) حتى نحدد مكان نقطة الاتصال الجديدة. وعملية المرور بالقائمة لنحدد مكان النقطة الجديدة سوف تحتاج إلى إضافة عدد من المتغيرات (أو إلى إضافة مؤشرين بالتحديد) المؤشر pNodePrevious (أي مؤشر إلى نقطة الاتصال السابقة) و pNodeCurrent (أي مؤشر إلى نقطة الاتصال الحالية) وكذلك مؤشر لنقطة الاتصال الجديدة كما يلي :

```
NODE * pNodeCurrent;
NODE * pNodeNew;
NODE * pNodePrevious;
```

سوف نقوم الآن بوضع المؤشر pNodePrevious لقيمة فارغة (NULL) وذلك لكي يدل على عدم وجود أي نقاط اتصال قبل نقطة الاتصال الأولى. والمؤشر

pNodeCurrent الذي سوف يكون متوجها دائما إلى نقطة الاتصال الحالية يحمل قيمة المؤشر pHead (المؤشر pHead يكون دائما متجها إلى نقطة الاتصال الأولى في القائمة) كما يلي :

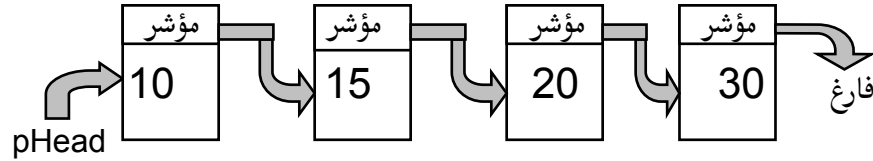
```
NODE* pNodePrevious = NULL;
NODE* pNodeCurrent = pHead;
```

وعندما نقوم بالمرور بالقائمة بداية بنقطة الاتصال الأولى في القائمة فإننا نقوم بمقارنة قيمة نقطة الاتصال الحالية بالتحديد pNodeCurrent->iValue (لا تنس أن نقاط الاتصال هي عبارة عن متغير من نوع typedef struct وتحمل متغيرين : متغير القيمة "iValue" ومتغير مؤشر إلى نقطة الاتصال التالية "pNodeNext") مع نقطة الاتصال الجديدة بالتحديد pNodeNew->iValue ، فإذا كانت قيمة نقطة الاتصال الحالية أكبر من نقطة الاتصال الجديدة ننتقل إلى النقطة التالية في القائمة (كيف ننتقل من نقطة إلى أخرى في القائمة؟ نقوم بذلك أولا عن طريق وضع المؤشر إلى النقطة السابقة "pNodePrevious" يحمل قيمة المؤشر إلى النقطة الحالية "pNodeCurrent" ثم وضع قيمة المؤشر إلى النقطة الحالية "pNodeCurrent" يساوي قيمة المؤشر إلى النقطة التالية pNodeCurrent->pNodeNext - أذكر مرة أخرى بعدم نسيان أن كل نقطة اتصال هي عبارة عن متغير من نوع typedef struct وتحمل متغيرين : متغير القيمة "iValue" ومتغير مؤشر إلى نقطة الاتصال التالية ("pNodeNext"

أما إذا كانت قيمة نقطة الاتصال الجديدة أقل من قيمة النقطة الحالية فإننا قد وجدنا المكان وعلينا أن نقوم بترتيب بعض المتغيرات حتى نستطيع إضافة النقطة الجديدة في القائمة. وسوف نرى الآن فائدة المؤشرين pNodePrevious و pNodeCurrent في مساعدتنا في هذه العملية.

قد يكون المؤشر `pNodePrevious` ذو قيمة فارغة "NULL" ، مما يدل على أن النقطة الجديدة سوف تكون نقطة الاتصال الأولى في القائمة في هذه الحالة نقوم فقط بوضع قيمة المؤشر `pNodeNew->pNodeNext` (هذا هو مؤشر نقطة الاتصال الجديدة إلى النقطة التالية) تساوي قيمة المؤشر `pHead` ثم نقوم بتغيير قيمة المؤشر `pHead` إلى `pNodeNew`.

إذا لم يكن المؤشر `pNodePrevious` ذو قيمة فارغة "NULL" ، إذا علينا أن نضع قيمة المؤشر `pNodeNew->pNodeNext` تساوي قيمة المؤشر `pNodePrevious->pNodeNext` وقيمة المؤشر `pNodeNew->pNodeNext` تساوي `pNodeNew`.



قد تجد نفسك أنك لازلتم لم تضيف نقطة الاتصال الجديدة والمؤشر `pNodeCurrent` له قيمة فارغة "NULL". هذا يعني أن نقطة الاتصال الجديدة يجب أن توضع في آخر القائمة. وعملية الإضافة في هذه الحالة هي نفسها كما لو كانت النقطة الجديدة ستوضع في الوسط. أولاً نقوم بوضع المؤشر `pNodeNew->pNodeNext` يساوي `pNodePrevious->pNodeNext` (في هذه الحالة، `pNodePrevious->pNodeNext` مؤشر له قيمة فارغة "NULL" لأن `pNodePrevious` كان آخر نقطة اتصال في القائمة). ثانياً نقوم بوضع قيمة المؤشر `pNodePrevious->pNodeNext`

pNodeNext > تساوي نقطة الاتصال pNodeNew وبذلك نكون قد أنهينا إضافة نقطة الاتصال الجديدة لأخر القائمة.

دعونا الآن ننظر إلي مثال –جميع أمثلة الكتاب توجد على القرص المدمج مع الكتاب– يقوم بما سبق وشرحناه. (هذا المثال مبني على لغة السي ونظام برمجة Dos حتى يكون سهل الفهم ويمكن استخدام نفس البرنامج مع نظام windows مع بعض الإضافات الخاصة ببرمجة النظام windows ، راجع برمجة win32 لتعرف تلك الإضافات):

سنقوم في بداية البرنامج بإنشاء متغيرين من نوع struct (المتغير NODE والمتغير LinkedList) كما يلي :

```
#include <stdio.h>
#include <stdlib.h>

typedef struct _node
{
    int iValue;
    struct _node * pNodeNext;
} NODE;

typedef struct _linkedlist
{
    NODE *      pNodeHead;
    int        cElements;
} LinkedList;
```

بعد ذلك نقوم بإضافة المتغيرات المطلوب استخدامها كمتغيرات عامة (Global variables) في البرنامج كما يلي :

```
#ifndef BOOL
#define BOOL int
#define TRUE 1
#define FALSE 0
#endif
```

ثم نقوم بتعريف كل الوظائف "functions" المستخدمة في البرنامج كما يلي:

```
BOOL FAddLinkedListElement(LinkedList *, int);
void PrintListElements(LinkedList *);
void RemoveAllElementsFromList(LinkedList *);
```

نقوم بعد ذلك بإنشاء الجزء الرئيسي من البرنامج (الوظيفة main):

```
int main (int argc, char ** argv)
{
    LinkedList List;
    int iTestValuesArray[5] = { 10, 20, 30, 5, 40 };
    int iTestValuesIndex;
```

نقوم بإعطاء القيم المبدئية للقائمة:

```
List.pNodeHead = NULL;
List.cElements = 0;
```

الآن سوف نقوم بإضافة بعض القيم للقائمة:

```
for (iTestValuesIndex = 0; iTestValuesIndex < 5;
    iTestValuesIndex++)
    {
        if (!FAddLinkedListElement(&List,
            iTestValuesArray[iTestValuesIndex]))
            {
                fprintf(stderr, "Error adding a node
to the linked list\n");
                exit(1);
            }
    }
```

نقوم بطبع القيم الموجودة في القائمة:

```
PrintListElements (&List);
```

الآن نقوم بتحرير جميع الذاكرة المحجوزة:

```
RemoveAllElementsFromList (&List);
```

```
return (0);
```

```
}
```

في هذا الجزء سوف نقوم ببرمجة وظيفة `FAddLinkedListElement` وتقوم هذه الوظيفة بإضافة عنصر إلى القائمة له قيمة من نوع `int` وكذلك إضافة `Node` إلى آخر القائمة. وتعيد صحيح (`TRUE`) عند النجاح و غير صحيح (`FALSE`) عند الفشل:

```

BOOL FAddLinkedListElement(LinkedList * pList , int
iValue)
{
    NODE * pNodeCurrent;
    NODE * pNodeNew;
    NODE * pNodePrevious;
    هنا نقوم بإضافة نقطة اتصال (node) جديدة :

    pNodeNew = (NODE *)malloc(sizeof(NODE));

    if (pNodeNew == NULL)
        return (FALSE);
    نقوم الآن بإعطاء المؤشر لنقطة الاتصال الجديد قيمة فارغة (NULL) وكذلك نعطي
    القيمة iValue لنقطة الاتصال:

    pNodeNew->pNodeNext = NULL;
    pNodeNew->iValue = iValue;

    الآن نقوم بتجهيز القائمة لعملية المسح (أو البحث):

    pNodePrevious = NULL;
    pNodeCurrent = pList->pNodeHead;

    نقوم الآن بعملية المسح وذلك لوضع نقطة الاتصال الجديدة في آخر القائمة:

    while (pNodeCurrent != NULL)
    {
        pNodePrevious = pNodeCurrent;
        pNodeCurrent = pNodeCurrent->pNodeNext;
    }

```

في هذه النقطة من البرنامج تكون نقطة الاتصال pNodePrevious تؤشر إلى المكان في القائمة الذي يسبق نقطة الاتصال الجديدة:

```
if (pNodePrevious != NULL)
{
    نقوم الآن بتعديل الاتصال بين النقطة السابقة والنقطة الجديدة:

    pNodeNew->pNodeNext = pNodePrevious->pNodeNext;
    pNodePrevious->pNodeNext = pNodeNew;
}
else
{
    إذا هذه نقطة الاتصال الأولى في القائمة:
    pList->pNodeHead = pNodeNew;
    لقد سبق وقمنا بإعطاء pNodeNew->pNodeNext قيمة فارغة (NULL) لذلك ليست
    هناك حاجة للتغيير هنا:
}
}
```

```
pList->cElements++; // Increase our count of
//elements in the list

return(TRUE);

}
```

نقوم هنا ببرمجة وظيفة PrintListElements وتقوم هذه الوظيفة بطبع جميع قيم قائمة الاتصال:

```
void PrintListElements(LinkedList * pList)
{
    NODE *      pNodeCurrent;
    int         iElement = 0;

    pNodeCurrent = pList->pNodeHead;

    printf("The linked list has %d nodes.\n\n", pList->cElements);
}
```

نقوم بعملية مسح حتى تكون نقطة الاتصال الجديدة هي آخر نقطة اتصال في

القائمة:

```
while (pNodeCurrent != NULL)
{
    printf("%d", pNodeCurrent->iValue);

    if (++iElement < pList->cElements)
    {
        printf(", ");
    }
    else
    {
        printf(".\n");
    }

    pNodeCurrent = pNodeCurrent->pNodeNext;
}
}
```

ونقوم أخيرا ببرمجة وظيفة RemoveAllElementsFromList وتقوم هذه الوظيفة بإزالة كل نقاط الاتصال من القائمة:

```
void RemoveAllElementsFromList(LinkedList * pList)
{
    NODE *      pNodeKill;
    NODE *      pNodeCurrent;

    pNodeKill   = NULL;
    pNodeCurrent = pList->pNodeHead;
}
```

نقوم بمسح القائمة بحيث نحرر كل نقطة اتصال على حدا:

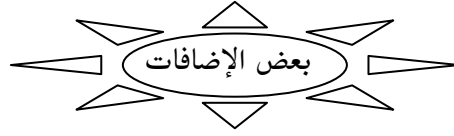
```
while (pNodeCurrent != NULL)
{
    pNodeKill      = pNodeCurrent;
    pNodeCurrent   = pNodeCurrent->pNodeNext;
    free(pNodeKill);
}
}
```

ثم نقوم بإعطاء قيم جديدة لكل من المتغيرين:

```
pList->pNodeHead = NULL;
pList->cElements = 0;

}
```

ويوجد البرنامج كاملا على القرص المدمج المرفق مع الكتاب.



لكي نجعل القائمة المتصلة في حالتنا مفيدة فإننا نحتاج إلى بعض الإضافات. فإذا صممنا القائمة بحيث تكون فعالة فإننا نستطيع استخدامها لبناء أنواع مختلفة من المعلومات المركبة (Data Structures). ولكن ما هي الإضافات المطلوبة ؟ في البداية سوف ننظر إلى مثالنا السابق ، سوف نلاحظ أننا استخدمنا أربع وظائف (functions) مختلفة لإضافة نقطة اتصال إلى القائمة : أولا في بداية القائمة ثانيا في نهاية القائمة ثالثا بشكل تصاعدي ورابعا بشكل تنازلي. ولو نظرنا إلى تلك الأوامر فإننا سوف نلاحظ الكثير من التشابه في أجزاءها مما يعطينا فكرة لوضعها في أمر واحد فقط. دعونا ننظر أولا إلى تلك الأجزاء المتشابهة ثم إلى الأجزاء المختلفة. جميع الأوامر متشابهة في طريقة حجز الذاكرة لنقطة الاتصال الجديدة وجميع الأوامر تتطلب معرفة مكان إضافة تلك النقطة الجديدة في القائمة وأجراء التغييرات المطلوبة للقيام بتلك العملية. ونرى الاختلاف فقط في طريقة معرفة مكان الإضافة.

وقد تكون لاحظت في المثال السابق أنه عند إنشاء القائمة المتصلة فإننا دائما نستخدم نفس الأمر لإضافة نقاط جديدة إلى القائمة. ونستطيع استغلال هذه الملاحظة في إضافة أمر اختيار لقائمة الاتصال وبحيث نستخدم متغير من نوع

integer نقوم من خلاله بتخزين عدد من البت (bits) تمثل الاختيارات المطلوبة. الآن سوف نقوم فقط بإضافة الاختيارات المطلوبة لنوعية الإضافة في القائمة. هذه الاختيارات سوف تكون في بداية القائمة وعندما نستدعي أمر الإضافة نقوم ببساطة في النظر إلى القائمة لتحديد نوعية الاختيار المطلوب للبحث.

```
typedef struct LinkedList
{
    NODE * pNodeHead;
    int cElements;
    int grbitOptions;
} LinkedList;
```

And for the bit flag definitions and bitmask for the insert option:

```
#define bitFInsertAtHead      0x0001
#define bitFInsertAtTail     0x0002
#define bitFInsertAscending  0x0004
#define bitFInsertDescending 0x0008
#define grbitInsertMask      0x000F
```

سوف نحتاج في القائمة متغير الاختيار (grbitOptions) ويجب أن يكون الاختيار المطلوب قد تم قبل استخدام هذا المتغير لذلك سوف نقوم ببرمجة أمر جديد FcreateLinkedList قبل البدء في استخدام القائمة المتصلة ويقوم هذا الأمر بأخذ متغيرين، المتغير الأول هو عبارة عن مؤشر إلى قائمة الاتصال (LinkedList) والثاني عبارة عن integer للاختيار المطلوب. ونستطيع أن نقوم باختيارين في نفس الوقت عن طريق الأمر المنطقي "أو" (OR). في الوقت الحالي سوف نستخدم اختيار واحد فقط وهو نوع الإضافة إلى القائمة ولكن قد نستخدم في المستقبل أكثر من اختيار مثلا لو أردنا القائمة أن تحتوي على نوع معين من النقاط كمتغير من نوع "strings" فيمكننا إضافة اختيار بحيث تكون النقاط الجديدة فقط من نوع الأحرف الصغيرة (تتكون الأحرف في اللغة الإنجليزية

من أحرف كبيرة "Capital letter" أو أحرف صغيرة "Small letter" يطلق عليها strings (strings) وهكذا.

ونستخدم الأمر الجديد "FcreateLinkedList" مثلا لإضافة نقطة اتصال جديدة بشكل تصاعدي كما يلي :

```
FCreateLinkedList(&TestList, bitFInsertAscending);
```

ويقوم الأمر FAddLinkedListElement بتحديد قيمة المتغير grbitOptions في القائمة المتصلة لكي يحدد نوع الاختيار المطلوب لإيجاد موقع نقطة الاتصال الجديدة. ويستخدم grbitInsertMask لإلغاء كل الاختيارات ماعدا الاختيار المطلوب.

الآن سوف نضيف وظيفة (function) تقوم بإعطاء القيم المبدئية لنقاط القائمة. هذه وظيفة تقوم ببساطة بوضع قيم معروفة. ففي القائمة نقوم بوضع قيمة المؤشر pNodeHead إلى قيمة فارغة (NULL) ونضع أصفار في الأماكن الباقية.

الآن لدينا وظيفة لإضافة نقاط الاتصال ووظيفة أخرى لإزالة جميع نقاط الاتصال. وكل ما تحتاج فعله عزيزي القارئ أن كنت قد استوعبت الشرح السابق هو أن تقوم بكتابة وظيفة للإطلاع على قيمة معينة في القائمة أو إزالة نقطة اتصال بمعرفة المكان أو القيمة. لأن هذه الوظائف ضرورية لتكون القائمة المتصلة مفيدة.

الفصل الثاني عشر

العنصر

DirectDraw

DIRECTDRAW

هذا هو العنصر الأساسي لـ DirectX لأنه العنصر المهتم بذاكرة الفيديو وكما نعرف بأن برامج الميلتي ميديا أو برامج الصوت والصورة و الألعاب تتعامل مع ذاكرة الفيديو بشكل رئيسي لأنها المهتمة بما يخرج على الشاشة من مشاهد ونستطيع أن نتصور DirectDraw كآلة العرض السينمائي لأنها المسئولة عن ما يظهر على الشاشة، طبعاً يتم ذلك من خلال خطوات سوف نراها بالتفصيل ومن التجربة سوف تعرف بأن فهم طريقة عمل هذا العنصر سوف توفر عليك الكثير من التساؤل عند التعامل مع العناصر الأخرى من عناصر DirectX وخاصة Direct3D لأنه يعتمد على DirectDraw في عملة والحقيقة أن طريقة إعداد هذا العنصر بسيطة (لا أقول ذلك لقصد التشجيع لأنها فعلاً بسيطة) ولكنها طويلة شيئاً ما وذلك لأن فقط إعداد برنامج يُظهر نافذة على الشاشة تحت نظام ويندوز طويل بالنسبة لنظام Dos (لاحظ أن هذا البرنامج لا يعمل شيئاً سوى إظهار نافذة على الشاشة) الشيء الإيجابي في هذه المسألة هو أنك لا تحتاج إلي معرفة ذلك إلا مره واحدة وتعيده مع كل برامج ويندوز ولهذا السبب ولجعل ذلك أكثر بساطة على المبرمج فإنه يأتي مع Visual C++ 6 برنامج يسمى AppWizard يقوم بتلك العملية بشكل تلقائي حتى نبدأ في البرمجة في الحال دون أن نقوم بنفس العملية كل مرة نريد فيها أن نبرمج

على نظام ويندوز ولكن نفس المسألة تنطبق على DirectDraw فطريقة إعدادها متشابهة في كل البرامج لذلك لماذا لا نستطيع أن نستخدم AppWizard معها كذلك. للأسف لا يقوم AppWizard بذلك لنا ولكن Visual C++ 6 يعطينا القدرة على تصميم AppWizard خاص بمتطلباتنا إذا أردنا ذلك. وقد وجدت من خلال التجربة بأن الاعتماد على AppWizard لإنتاج هيكل لبرنامجنا يفقدنا القدرة على معرفة كيفية عمل تكنولوجيا DirectDraw ولذلك لن نستخدم AppWizard في الوقت الحالي وسوف نقوم بشرح كل الخطوات اللازمة لإعداد نافذة على الشاشة (كما شرحناها في الفصل الثالث من هذا الكتاب ، برمجة win 32) ثم إعداد هذا العنصر من عناصر DirectX.

ما هي DirectDraw ؟

قبل أن نبدأ بشرح DirectDraw دعونا نضيفها إلى برنامج win32 حتى نرى سهولة أضافتها واستخدامها. قد تعتقد من الوهلة الأولى (كما كنت أعتقد أنا) أن هناك الكثير لتعلمه في كل مرة تتعرف فيها على شيء جديد خاصة وأن لم يكن عندك أي معرفة ببرمجة win32 وكما أحب أن أطمئنك بأن كل البرامج التي تستخدم DirectX تقوم بنفس الخطوات لذلك ليس عليك إلا تعلم win32 مرة واحدة فقط بحيث تستخدمها كلما قمت باستخدام DirectX. في البداية سوف

نضيف بعض تعاريف المتغيرات (لا تنسى طبعا أن تضيف ddraw.lib إلى قائمة Link تحت Project Settings....) التي سوف نستخدمها كما يلي :

```
#include <ddraw.h>
```

```

HWND hwnd; //Windows program handle
static short *pBase;

// Direct draw globals

LPDIRECTDRAWSURFACE lpDDSPPrimary; // DirectDraw
// primary surface
LPDIRECTDRAWSURFACE lpDDSPBack; // DirectDraw back
// surface
BOOL bActive; // is application active?

BOOL DirectDrawInit();

```

أولا نقوم بنقل التعريف **HWND hwnd** من الخطوة الأولى إلى أعلى البرنامج

```
bActive=DirectDrawInit ();
```

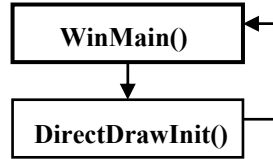
ثالثا نقوم بإضافة **{ DirectDrawInit ()** إلى آخر البرنامج

هذا كل ما في الأمر (سهل أليس كذلك 😊) في هذه النقطة استطعنا أن نحول برنامج **win32** ليقوم باستخدام العنصر **DirectDraw** للتعامل مع ذاكرة الفيديو. ونرى ذلك من خلال **DirectDrawInit()**; وهو الجزء الذي أضفناه لإعداد البرنامج لاستخدام هذا العنصر. من خلاله قمنا بعدة خطوات سوف نشرحها عن طريقها

استطعنا أن نجعل DirectDraw جزء من البرنامج (أنظر إلى الشكل (1)) ونرى

هذا

الجزء كما يلي:



الشكل (1)

```

BOOL DirectDrawInit ( void ) {
    LPDIRECTDRAW lpDD;
    HRESULT ddrval;

    ddrval = DirectDrawCreate ( NULL, &lpDD, NULL);
    if ( ddrval != DD_OK) return FALSE;

    ddrval = IDirectDraw_SetCooperativeLevel ( lpDD, hwnd, DDSCCL_EXCLUSIVE |
    DDSCCL_FULLSCREEN);
    if ( ddrval != DD_OK) return FALSE;

    ddrval = IDirectDraw_SetDisplayMode ( lpDD, 640, 480, 16);
    if ( ddrval != DD_OK) return FALSE;

    // create foreground surface
    DDSURFACEDESC ddsd;

    ddsd.dwSize = sizeof(DDSURFACEDESC);
    ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
    ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE | DDSCAPS_FLIP |
    DDSCAPS_COMPLEX;
    ddsd.dwBackBufferCount = 1;
    ddrval = IDirectDraw_CreateSurface ( lpDD, &ddsd, &lpDDSPPrimary, NULL);
    if ( ddrval != DD_OK) return FALSE;

    // create buffer surface
    ddsd.dwFlags=DDSD_CAPS|DDSD_HEIGHT|DDSD_WIDTH;
    ddsd.ddsCaps.dwCaps=DDSCAPS_OFFSCREENPLAIN|DDSCAPS_SYSTEMMEMORY;
    ddsd.dwHeight=480;
    ddsd.dwWidth=640;
    ddrval=IDirectDraw_CreateSurface( lpDD, &ddsd, &lpDDSDBack, NULL );
    if ( ddrval != DD_OK) return FALSE;

    // Get pointer to buffer surface
    IDirectDrawSurface_Lock(lpDDSDBack,NULL, &ddsd, 0, NULL);
    pBase = ( ddrval==DD_OK ? (short *)ddsd.lpSurface : NULL);
  
```

```

IDirectDrawSurface_Unlock(lpDDSDBack, NULL);
return TRUE;
}

```

ولكي تكون الأمور واضحة يجب أن نشرح أولاً طريقة عمل DirectDraw. ولكي

نستخدم هذا العنصر في برنامجنا يجب أن نقوم بالخطوات التالية :

1. إعداد وتعريف المتغيرات المستخدمة بشكل عام (global variables)

2. بداية إنشاء كائن DirectDraw

3. إعداد مستوى العرض "cooperative level"

4. إنشاء سطح أمامي (Front Surface) و سطح (أو أكثر) خلفي (Back Surface)

لذاكرة الفيديو وكائن حاذف (clipper object) إذا احتجنا لذلك.

5. نلحق الكائن الحاذف بالسطح الرئيسي لذاكرة (primary surface)، ولا

نحتاج لإنشاء هذا السطح لأنه ينشأ بشكل تلقائي عند إنشاء كائن

DirectDraw الفيديو إذا كنا نستخدم نافذة بدل الشاشة بكاملها.

6. نقوم بقلب سطحي الذاكرة، في حالة استخدام الشاشة بكاملها نقوم فقط

بقلب الأسطح باستخدام الأمر flip ولكن في حالة النافذة نقوم باستخدام الأمر

blit الذي يقوم بنسخ محتويات السطح الخلفي إلى السطح الرئيسي في كل

إطار.

الآن كيف طبقنا هذه الخطوات في برنامجنا السابق؟

في الخطوة (1) قمنا بتعريف المتغيرات المستخدمة (راجع البرنامج لترى

المقصود) ، المتغير lpDDSPPrimary وهو متغير السطح الرئيسي والظاهر لذاكرة

الفيديو ثم المتغير lpDDSDBack للسطح الخلفي للذاكرة وفي DirectDrawInit

(هي الجزء من البرنامج المهتم بإعداد DirectDraw) قمنا بتعريف المؤشر lpDD

لـ DirectDraw (مؤشر بمعنى Pointer) ثم المتغير ddrval وهو المتغير

المستخدم للتعرف على نجاح كل خطوة نقوم بها لإنشاء DirectDraw فيما إذا فشلت خطوة يقوم هذا المتغير بإبلاغنا.

في الخطوة (2) قمنا باستخدام الأمر DirectDrawCreate وذلك لإنشاء كائن DirectDraw ولاحظ أننا جعلناه يساوي المتغير ddrval وذلك كما سبق وأشرنا أنه يقوم بتسجيل نتيجة الخطوة أن كانت ناجحة فإنه سوف يساوي واحد وأن كانت فاشلة فإنه سوف يساوي صفر ونقوم بذلك مع بقية الأوامر.

في الخطوة (3) بعد أن إنشائنا الكائن المطلوب قمنا باستخدام الأمر IDirectDraw_SetCooperativeLevel ويقوم هذا الأمر بإعداد مستوى العرض بحيث يكون في هذه الحالة DDSCL_FULLSCREEN أي أن DirectDraw سوف تستغل الشاشة بأكملها بعد أن قمنا بتحديد شكل برنامجنا (نافذة أو الشاشة بأكملها) قمنا باستخدام أمر تحديد نوعية العرض وهو الأمر IDirectDraw_SetDisplayMode سواء كان من حيث دقة الألوان (8 بت أو 16 بت أو 24 بت) أو حجم الشاشة في هذه الحالة 480X640.

في الخطوة (4) قمنا أولاً بإنشاء السطح الأمامي عن طريق الأمر IDirectDraw_CreatSurface() (وهو السطح الرئيسي في هذه الحالة) ثم استخدمنا نفس الأمر مرة أخرى لإنشاء السطح الخلفي.

في الخطوة (5) لم نفعل شيء لأننا نتعامل مع الشاشة بأكملها.

في الخطوة (6) وعن طريق الوظيفة DrawStuff() قلبنا محتويات السطح الخلفي إلى السطح الرئيسي عن طريق الأمر IDirectDrawSurface_BltFast() .

وهذا كل ما في الأمر عند استخدام العنصر DirectDraw. طبعاً هناك بعض التفاصيل التي لم أرى أهميتها في هذه النقطة من تعرفنا على هذا العنصر وسوف نتطلع عليها لاحقاً (البرنامج موجود في القرص المدمج في الملف win32+Ddraw).

ولكن هذا البرنامج لا يقوم بشي مختلف عن السابق ما عدا تغيير الشاشة إلى 480x640 وحتى نرى فعلا قوه هذا العنصر دعونا نقوم ببعض الإضافات للبرنامج ليقوم بشيء معين (سوف يقوم برنامجنا برسم ألوان مختلفة على الشاشة وتحريكها).

أولا نقوم بإضافة ما يلي إلى أعلى البرنامج

```
static int addColor=0; // متغير للألوان
void DrawStuff(); // جزء آخر من البرنامج مهتم في رسم الألوان
```

ثانياً في الخطوة السابعة

نقوم بتغيير دورة الرسائل إلى الشكل التالي:

```
while (TRUE) { // Message loop
    if( PeekMessage( &msg, NULL, 0, 0, PM_NOREMOVE ) ) {
        if( !GetMessage( &msg, NULL, 0, 0 ) )
            return msg.wParam;
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    } else if (bActive) DrawStuff();
}
```

ثالثاً

نضيف الوظيفة DrawStuff() إلى اسفل البرنامج كما يلي:

```
void DrawStuff() {
    short *pBuff=pBase;

    if (pBuff) {
        RECT r={0,0,640,480};

        // paint something
    }
}
```

```

int i, j;
for (j=0; j<480; j++)
for (i=0; i<640; i++, pBuff++)
*pBuff=i+j+addColor;

addColor+=(1 << 11) + (1 << 5) + 1;

// Update the screen ...

IDirectDrawSurface_BltFast(lpDDSPPrimary,0,0,lpDDSBack,
&r,DBBLTFAST_NOCOLORKEY);

}
}

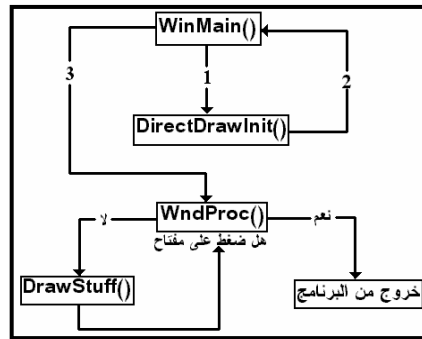
```

بعد إتمام هذا الخطوة فإن برنامجنا سيقوم بعملية بشكل كامل ولكن بقي هناك أمر بسيط يجب القيام به في الخطوتين التاليتين:

نقوم بإزالة الخطوة التاسعة (عن طريق مسح كل محتوياتها)

وأخيراً في الخطوة العاشرة

نقوم بإزالة case WM_PAINT وكل محتوياتها هذا كل ما في الأمر (البرنامج موجود في القرص المدمج في الملف win32+Ddraw2)، قمنا بإزالة جزء من البرنامج لعدم حاجتنا إليه، أنت تتساءل عن معنى الخطوات التي قمنا بها طبعاً. الشكل التالي يوضح طريقة عمل البرنامج:



كل ما قمنا به هو إضافة وظيفة (`DrawStuff`) والتي تقوم ببساطة برسم وتحريك ألوان معينه على السطح الخلفي ثم تستخدم الأمر `BltFast` من أوامر `DirectDraw` لتحويل محتويات السطح الخلفي إلى السطح الرئيسي:

```
IDirectDrawSurface_BltFast(lpDDSPrimary,0,0,lpDDSBack,
&r,DDBLTFAST_NOCOLORKEY);
```

الرئيسي \longrightarrow هذا المتغير يمثل السطح `LpDDSPrimary`
 الخلفي \longrightarrow هذا المتغير يمثل السطح `LpDDSBack`
 \longrightarrow هذا متغير من نوع `struct` يحتوي على مواصفات حجم الشاشة.

`DDBLTFAST_NOCOLORKEY` أما هذا فيمثل الاختيار بتحويل الصورة على السطح الخلفي إلى السطح الرئيسي بدون الأخذ في الاعتبار أي لون شفاف. وتتكرر هذه العملية مرات عديدة في الثانية فنرى حركة الألوان.

دعونا أخيرا نقوم بتغيير بسيط في البرنامج السابق بحيث نقوم بنقل جميع التعاريف المستخدمة في البرنامج والتي كانت معرفة قبل (`WinMain`) إلى ملف `win32.H` تعريف خاص بها نسميه `win32.H` بعد ذلك نقوم بإضافة الأمر `#include "win32.h"` إلى أعلى البرنامج (يوجد في القرص المدمج مع الكتاب في الملف `win32+Ddraw3`).



يا إلهي ، حجم البرنامج يزداد كثيرا كلما أضفنا شيء جديد هل هناك

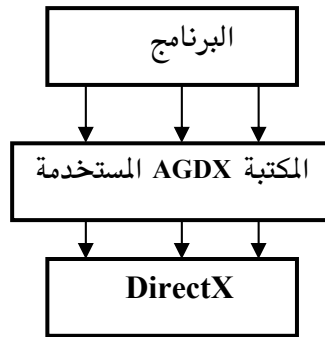
حل؟؟

هذا صحيح ، فحجم برامج الصوت والصورة يزداد بشكل ملاحظ عند إضافة بعض المميزات من وقت لآخر. الأمر المفيد هنا أن اغلب الإضافات ثابت في كل البرامج ولا يحتاج إلى أي تغيير ولذلك سوف نستخدم الكائنات (Classes) في تخزين الإضافات فذلك سوف يسهل علينا الأمور. مثلا عرفنا أن DirectDraw تتحكم في الشاشة وكذلك عرفنا من البرنامج السابق أن DirectDraw قامت بإنشاء واستخدام سطحين ، السطح الرئيسي المرئي على الشاشة ثم السطح الخلفي الغير مرئي للمستخدم. لذلك لجعل الأمور أكثر سهولة سوف نقوم بنفس العملية التي قمنا بها في البرنامج السابق ولكن الآن عن طريق إنشاء كائنين الأول هو "Screen" و المهتم بما يظهر على الشاشة الرئيسية ثم بعد ذلك سوف نستخدم كائن آخر نطلق عليه "Surface" وهو الكائن المهتم بإنشاء الأسطح على ذاكرة الفيديو والتحكم بها. عن طريق هذين الكائنين سوف تصبح الأمور أكثر سهولة ، وما علينا كمبرمجين سوى طلب الكائن من البرنامج الرئيسي فيقوم بعمله بدون الحاجة إلى رؤيته (بهذه الطريقة سوف نقوم بإنشاء كائنات مكتبة الألعاب المستخدمة في هذا الكتاب ثم بعد ذلك سوف نقوم بوضع جميع تلك الكائنات سواء كائن الصوت أو كائن الشاشة ... الخ في مكتبة Lib بحيث لا نحتاج بعد ذلك للنظر إليها وإنما نقوم بالتركيز على طريقة عمل برنامجنا كما كان الحال في الجزء الأول من الكتاب).



عذراً ولكن لم أفهم ما تقصد؟؟

لا بأس (هذا هو السبب الرئيسي الذي جعلك تشتري الكتاب، لكي تكون الأمور واضحة) نحن نستخدم مكتبة لإنشاء البرامج التي نتعامل معها (سواء كانت برامج ألعاب أو كل ما يستخدم تكنولوجيا دايركت أكس "DirectX" وهذه المكتبة عبارة عن كائنات مختلفة لكل منها وظيفة معينة سوف نتعرف عليها. بعد أن نقوم بإنشاء هذه الكائنات المختلفة لن تكون هناك حاجة لرؤيتها والسبب في ذلك أننا لن نحتاج إلى تغييرها مهما اختلف البرنامج (في الحقيقة يمكن تغييرها إذا شأنا ولذلك خصصت هذا الجزء من الكتاب لشرح كيفية إنشاء المكتبة والتي بدورها تستخدم هذه التكنولوجيا) ولكي يكون برنامجنا سهل الفهم سوف نقوم بوضع جميع هذا الكائنات في تلك المكتبة. يتم ذلك طبعاً عن طريق استخدام Microsoft Visual C++ لإنتاج ملف واحد من نوع lib نستخدمه في برنامجنا بدون أن نقلق بشأن هذه الكائنات بعد ذلك.



العنصر DirectDraw-المثال الثاني

عزيزي القارئ بعد أن تعرفنا في المثال السابق على كيفية إضافة العنصر DirectDraw دعونا في هذا المثال نلقي الضوء مرة أخرى على نفس البرنامج تقريبا ولكن مع بعض التعديلات. أولا سوف نقوم باستغلال آخر إصدارات DirectX وهو الإصدار السادس وذلك عن طريق تعديل وظيفة إعداد **DirectDraw**. وبعد أن نتعرف على هذا العنصر سنتعرف كذلك على الأسطح وكيفية إنشائها. ثم بعد ذلك سوف نقوم بإضافة وظيفة الإعداد إلى برنامجنا في الجزء الأول (بعد أن تخلصنا من برمجة win32).

وظيفة إعداد DirectDraw وأسطحها

```

BOOL DirectDrawInit(int Width, int Height, int BitDepth)
{
    LPDIRECTDRAW lpDD;
    HRESULT ddrval;

    ddrval = DirectDrawCreate ( NULL, &lpDD, NULL);
    if (ddrval != DD_OK) return FALSE;

    // Retrieve the DirectDraw4 interface(newer).
    ddrval = lpDD->QueryInterface(IID_IDirectDraw4, (void **)&DD4);
    if (ddrval != DD_OK) return FALSE;

    lpDD->Release();

    ddrval = DD4->SetCooperativeLevel ( hWnd, DDSCL_EXCLUSIVE |
    DDSCL_FULLSCREEN);
    if (ddrval != DD_OK) return FALSE;

    ddrval = DD4->SetDisplayMode ( Width, Height, BitDepth, 0, 0);
    if (ddrval != DD_OK) return FALSE;

    // create foreground surface
    DDSURFACEDESC2 ddsd;
    DDSCAPS2 ddscaps;

    ZeroMemory(&ddsd, sizeof(ddsd));
    ddsd.dwSize = sizeof(ddsd);

    ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
    ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE | DDSCAPS_FLIP |
    DDSCAPS_COMPLEX;
    ddsd.dwBackBufferCount = 1;
    ddrval = DD4->CreateSurface (&ddsd, &lpDDS4Primary, NULL);
    if ( ddrval != DD_OK) return FALSE;

    // create buffer surface
    ddsd.dwFlags=DDSD_CAPS|DDSD_HEIGHT|DDSD_WIDTH;
    ddsd.ddsCaps.dwCaps=DDSCAPS_OFFSCREENPLAIN|DDSCAPS_SYSTEMMEMORY;
    ddsd.dwHeight=Height;
    ddsd.dwWidth=Width;
    ddrval=DD4->CreateSurface( &ddsd, &lpDDS4Back, NULL );
    if ( ddrval != DD_OK) return FALSE;

    // Get a pointer to the back buffer
    ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
    ddrval=lpDDS4Primary->GetAttachedSurface(&ddscaps, &lpDDS4Back);

    // Create a timer to flip the pages
    if (!SetTimer(hWnd, 1, 500, NULL))
        return FALSE;

    return TRUE;
}

```

أخذت **DirectDraw** في التطور من الإصدار الأول لتكنولوجيا **DirectX** إلى الإصدار الحالي (السادس) ولكن طبعا طريقة برمجتها لم تتغير وسوف نقوم في هذا الدرس بكتابة وظيفة خاصة بإعداد هذا العنصر وتهيئته للاستخدام (سنسميها **DirectDrawInit()** نراها في الصفحة السابقة) ثم وفي نفس الوظيفة سنقوم بإنشاء سطحين للشاشة : سطح رئيسي ظاهر للمشاهد ، و سطح خلفي خفي عن المشاهد. سنقوم أولاً بالنظر إلى الشفرة البرمجية لهذه الوظيفة في الصفحة السابقة بعد ذلك سنقوم بشرحها ولماذا نحتاج إلى أسطح للشاشة.

كما نرى أن وظيفة الإعداد طويلة شيئا ما ولكنها مثل باقي الوظائف الرئيسية لا تتغير من برنامج إلى آخر فمعرفة مهمتها لأنها العنصر الرئيسي في هذه التكنولوجيا. سوف نقوم بشرحها بشكل تسلسلي ، بمعنى أننا سنبدأ من السطر الأول في أعلى الوظيفة ونستمر إلى السطر الأخير في أسفلها.

- أولاً بالنظر إلى الوظيفة **DirectDrawInit()** نرى أنها تقوم بأخذ العرض (**Width**) و الطول (**Height**) وعدد الألوان (**BitDepth**) كمعطيات لها عند استدعائها من الوظيفة الرئيسية (**Main**) (كما سوف نرى في الوظيفة الرئيسية) وهذا يعني أن لدينا الآن ثلاث متغيرات يحملن ثلاث أرقام مختلفة كل رقم له معنى معين وهو طول الشاشة وعرضها وعدد الألوان بها (وبالتالي ستكون هذه موصفات السطح الرئيسي والسطح الخلفي لأنهما يمثلان الشاشة). وسوف نستخدم هذه المتغيرات لإعداد الشاشة بالشكل المطلوب كما سوف نرى.

هيا بنا ننشئ DirectDraw4

- نبدأ في إعداد العنصر كما يلي :

```
LPDIRECTDRAW lpDD;
HRESULT ddrval;

ddrval = DirectDrawCreate ( NULL, &lpDD, NULL);
if (ddrval != DD_OK) return FALSE;

// Retrieve the DirectDraw4 interface(newer).
ddrval = lpDD->QueryInterface(IID_IDirectDraw4, (void **)&DD4);
if (ddrval != DD_OK) return FALSE;

lpDD->Release();
```

1. أولاً قمنا بتعريف كائن (Object) من نوع DirectDraw الإصدار الأول أطلقنا عليه IpDD (اخترنا هذا الاسم لأنه اختصار جملة: مؤشر إلى DirectDraw).

2. في السطر الذي يليه قمنا بتعريف المتغير ddrval وهو عبارة عن متغير يحمل قيمة معينة ستكون أداه نتعرف عن طريقها إلى نوع الخطأ الذي حصل في حالة حدوثه.

3. في السطر الذي يليه :

نقوم بإنشاء العنصر DirectDraw عن طريق الأمر التالي :

```
DirectDrawCreate ( NULL, &lpDD, NULL)
```

وكما تلاحظ أن كل ما نحتاجه لأمر الإنشاء هو المؤشر IpDD مسبق بإشارة "&" الذي أنشأناه في الخطوة رقم 1.

1. في هذه الخطوة سنقوم بتحويل عملية الإنشاء من الإصدار الأول لهذا العنصر والذي عبرنا عنه بالمؤشر IpDD إلى الإصدار الرابع والذي عبرنا عنه (في أعلى

الملف) بالمؤشر DD4. ولكي نقوم بهذه العملية سوف نحتاج إلى أمر التغيير وهو QueryInterface() والذي قام بتغيير عملية الإنشاء كما هو مطلوب.

2. وأخيرا قمنا بتحرير (أي مسح) المؤشر lpDD عن طريق الأمر Release() من الذاكرة لأننا لم نعد في حاجة إليه ولذلك نريد أن نستعيد المقدار الذي حجزه من الذاكرة وسوف نستخدم بدله المؤشر DD4.



أنا المؤشر DD4 وأمثلة العنصر DirectDraw وسأكون مسؤول عن كل ما يحدث على الشاشة ولكي انفذ ما تطلبني أن افعله استخدم اسمي مع السهم المتجه لليمين " → "

بعدما اصبح لدينا كائن لـ DirectDraw والمؤشر DD4 كمثل له بإمكاننا استخدام مميزاته (عن طريق السهم المتجهة إلى اليمين → " كما سوف نرى) وهي :

أولا التحكم في الشاشة باستغلالها كاملة (طبعا نستطيع أن نضعها كنافذة من نوافذ نظام التشغيل كباقي البرامج ولكننا في هذه الدروس سوف نركز على استخدام الشاشة بكاملها) وللقيام بذلك استخدمنا الأمر التالي :

```
ddrval = DD4->SetCooperativeLevel ( hWnd, DDSCL_EXCLUSIVE | DDSCL_FULLSCREEN);
if (ddrval != DD_OK) return FALSE;
```

طبعا كما نرى ، يجب أن نقوم بالتأكد أن العملية تمت بنجاح عن طريق فحص قيمة المتغير `ddrval`

الآن بعد أن جعلنا الشاشة جاهزة للاستعمال ، أصبح بإمكاننا تغيير الشاشة إلى العمق النقطي المطلوب بطول معين = قيمة المتغير `Height` وعرض معين = قيمة المتغير `Width` (`640x480` هو العمق المستخدم لهذا المثال تستطيع تغيير ذلك ولكن يعتمد هذا على حسب بطاقة العرض المستخدمة) ثم بعد ذلك نريد أن نجعل عدد الألوان = قيمة المتغير `BitDepth` (سبق وتحدثنا عن هذه المتغيرات في البداية) ونقوم بذلك عن طريق هذا الأمر:

```
ddrval = DD4->SetDisplayMode ( Width, Height, BitDepth, 0, 0);
if (ddrval != DD_OK) return FALSE;
```

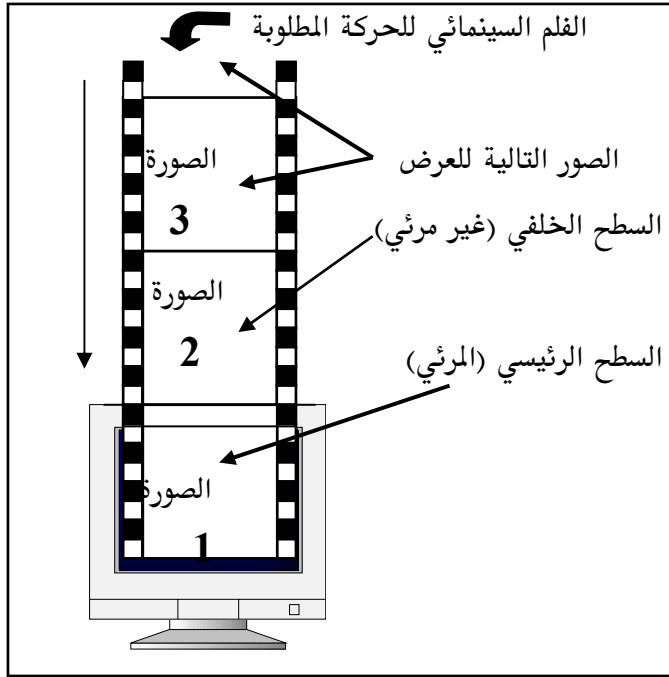
كما نرى طبعا، يجب أن نقوم بالتأكد أن العملية تمت بنجاح عن طريق فحص قيمة المتغير `ddrval`

”جميل جدا“ تقول ”وماذا الآن؟“ (أرجو أن لا يكون الملل دب إليك، عليك بالصبر وسيكون الموضوع بأكمله غاية في التسلية. صدقني)

السطح الرئيسي والسطح الخلفي

هل ذهبت يوماً إلى السينما ؟ حتى لو أنك لم تذهب إلى السينما لابد وأنك شاهدة الكثير من الأفلام والعروض على شاشة التلفاز ، ولكن هل خطر ببالك هذا السؤال "كيف استطاعوا أن يجعلوا من الصور أن تتحرك ؟" بمعنى آخر كيف تتكون الحركة ؟ ولكي نستطيع أن نبرمج الحركة على شاشة الكمبيوتر لابد لنا من التعرف على الحركة وكيفية إنشائها. إذا نظرت إلى صورة معينة سبق لك والتقطتها فإن تلك الصورة تمثل لحظه معينة ، ولو انك التقطت صور كثيرة الواحدة تلو الأخرى لشيء يتحرك ثم نظرت إلى تلك الصور بنفس السرعة والترتيب سوف ترى الحركة ، وهذه هي الطريقة المستخدمة للحركة في دور العرض (السينما). فالعلم عبارة عن صور عديدة تم التقاطها الواحدة تلو الأخرى وتعرض عن طريق أداة للعرض فتراها العين بشكل متكرر فتنشأ الحركة. والأسطح هنا هي أداة العرض التي سوف نستخدمها لتعرض الحركة على شاشة الكمبيوتر بنفس الأسلوب المتبع لإنشاء الحركة في دور العرض.

سوف نقوم بإنشاء سطح رئيسي مرئي للمشاهد يعرض الصورة الحالية وننشئ سطح خلفي آخر غير مرئي يحتفظ بالصورة التالية للعرض ، في كل جزء من الثانية نقوم بقلب محتويات السطح الخلفي إلى السطح الرئيسي حتى نرى الصورة التالية وفي نفس الوقت نأتي بصورة جديدة ونضعها في السطح الخلفي. بهذه الطريقة نستطيع إنتاج الحركة وبشكل مذهل. عملية سهلة أليس كذلك ؟ الآن تستطيع أن تحقق حلم طفولتك وتصبح مخرج أفلام على الكمبيوتر 😊.



شكل الأسطح وعلاقتها بالحركة

بعد أن تكونت لدينا فكرة ولو بسيطة عن الأسطح وسبب استخدامها لنرى الآن كيف يمكننا إنشائها ، وكما رأينا أن لكل سطح مساحة معينة بطول معين وعرض معين (تكون الأسطح عادة كلها بنفس المساحة كما في الفلم السينمائي ولكن هذا ليس شرطاً).

لننظر الآن كيف استطعنا أن ننشئ الأسطح المطلوبة ، وللقيام بذلك يجب أن نتبع الخطوات التالية :

- أولاً نقوم بإنشاء التعاريف المستخدمة كما يلي :

```
DDSURFACEDESC2 ddsd;
DDSCAPS2    ddscaps;
```

```
ZeroMemory(&ddsd, sizeof(ddsd));
```

ddsd.dwSize = sizeof(ddsd);

التعريف الأول "ddsd" هو ما يعرف DDSURFACEDESC2 وهو مجرد متغير من نوع Struct لا أكثر ولا أقل يحمل صفات السطح المطلوب إنشائه. طبعاً في هذه المرحلة لا يحمل أية صفات ولكن سنقوم بتعبئته في الخطوات التالية بالمعلومات المطلوبة (انظر إلى الجدول إذا أردت التعرف على الأعضاء) لكل سطح نريد إنشائه.

DDSURFACEDESC2

العضو	ماذا يعني (الشرح)
DWORD dwSize DWORD dwFlags	حجم المتغير ، يجب استخدام هذا العضو التحكم في الموصفات
DWORD dwHeight	ارتفاع السطح
DWORD dwWidth	عرض السطح
LONG lPitch	طبقة السطح
DWORD dwBackBufferCount	عدد الأسطح الخلفية
DWORD dwRefreshRate	عدد مرات تجديد الشاشة
DWORD dwAlphaBitDepth	هذا العضو ليس للاستعمال حالياً
DWORD dwReserved	هذا العضو ليس للاستعمال حالياً
LPVOID lpSurface	العضو المؤشر إلى هذا السطح
DDCOLORKEY ddckCKDestOverlay	مفتاح ألوان الهدف للـ overlays
DDCOLORKEY ddckCKDestBlit	مفتاح ألوان الهدف للـ blits
DDCOLORKEY ddckCKSrcOverlay	مفتاح ألوان المصدر للـ overlays
DDCOLORKEY ddckCKSrcBlit	مفتاح ألوان المصدر للـ blits
DDPIXELFORMAT ddpfPixelFormat	نوعية النقاط المستخدمة
DDSCAPS2 ddsCaps	مميزات السطح

جدول الأعضاء (لا تجعل هذا الجدول يخيفك استخدمه كمرجع فقط)

التعريف الثاني "ddscaps" وهو ما يعرف DDSCAPS2 وهو المهم بمميزات السطح (عن طريقة نستطيع استغلال بعض المميزات التي سنتعرف عليها في دروس قادمة) وهو أيضاً (لو نظرت إلى الجدول) أحد أعضاء المتغير DDSURFACEDESC2 ، وكان من الممكن استخدامه مباشرة كباقي الأعضاء ولكننا فضلنا استخدامه بشكل منفصل وذلك لتسهيل عملية التعامل معه.



سؤال : ماذا تقصد باستخدامه مباشرة كباقي الأعضاء ؟

لو نظرت إلى جدول أعضاء المتغير DDSURFACEDESC2 ستكتشف أن العضو الأخير من أعضائه هو DDSCAPS2 وبما أننا أنشأنا متغير من نوع DDSURFACEDESC2 فإننا نستطيع استخدام جميع الأعضاء الموجودة فيه ، وبذلك نستطيع أن نستخدم هذا العضو مباشرة بدون الحاجة إلى إنشائه مرة أخرى بشكل منفصل. يعني بالعربي الفصيح بإمكانك أن تحذف تعريف هذا المتغير "ddscaps" واستخدام التعريف "ddsCaps" بدل منه إذا أردت. لاحظ أن "dds" تمثل المتغير من نوع DDSURFACEDESC2 والنقطة تعني أننا سوف نستخدم العضو المعين (في هذه الحالة هو ddsCaps) وما يفصلهم هو النقطة "." (ملاحظة : سوف نستخدم هذا المؤشر بشكل منفصل في آخر الوظيفة)

تعبئة موصفات الأسطح (أعضاء المتغير "dds") :

بعد ذلك ولاستخدم الذاكرة المطلوبة للأسطح يجب :

1. استخدم الأمر (ZeroMemory(&dds, sizeof(dds))) ولاحظ أن كل ما احتاج له هذا الأمر لكي ينفذ مهمته هو التعريف "dds" وهذا الأمر هو المسؤول عن جعل الذاكرة خالية من أي معلومات سابقة.
2. تعبئة العضو dds.dwSize ليساوي الحجم المطلوب لتخزين المتغير "dds" من نوع DDSURFACEDESC2 كما يلي "dds.dwSize = sizeof(dds)".

الآن بعد أن استوفينا شروط الذاكرة نستطيع أن نكمل تعبئة بقية الأعضاء ، أولا سوف نبدأ بتعبئة الموصفات والمميزات المطلوبة لإنشاء السطح الرئيسي :

- تعبئة العضو dwFlags بالمواصفات المطلوبة للسطح الرئيسي ، وفي هذه الحالة كما يلي :

```
ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
```



- تعبئة العضو `ddsCaps` بالمميزات المطلوبة للسطح الرئيسي كما يلي :

```
ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE |  

DDSCAPS_FLIP | DDSCAPS_COMPLEX;
```

- تعبئة العضو `dwBackBufferCount` بعدد الأسطح الخلفية المطلوب إنشائها "1" في هذا المثال.

الآن نستطيع أن نقوم بإنشاء السطح الرئيسي بعد أن قمنا بتعبئة الأعضاء المطلوبة للمتغير "ddsd" ولإنشاء أي سطح نعود لممثل DirectDraw4 وهو DD4 ونطلب منه أن ينشئ السطح الرئيسي كما يلي:

```
ddrval = DD4->CreateSurface (&ddsd, &lpDDS4Primary, NULL);
if ( ddrval != DD_OK) return FALSE;
```

المؤشر إلى السطح الرئيسي والذي عرفناه في أول الملف.

طبعا نقوم بالتأكد أن العملية مرت بنجاح عن طريق اختبار قيمة المتغير " ddrval " وهل هي OK أم لا .

الآن هل نستطيع أن نأمر ممثل DirectDraw4 بأن يقوم بإنشاء السطح الخلفي بمعلومية أننا قد سبق لنا وعيّننا مميزات "ddsd" ؟ لا لماذا ؟ لأننا عيّننا المتغير "ddsd" بمميزات عن السطح الرئيسي وليس السطح الخلفي ويجب الآن أن نقوم بتعبئة بعض القيم المختلفة لتكون مناسبة للسطح الخلفي كما يلي:

```
ddsd.dwFlags=DDSD_CAPS|DDSD_HEIGHT|DDSD_WIDTH;
```

```
ddsd.ddsCaps.dwCaps=DDSCAPS_OFFSCREENPLAIN|DDSCAPS_SYSTEMMEMORY;
```

```
ddsd.dwHeight=Height;
```

```
ddsd.dwWidth=Width;
```

أعتقد أنك تلاحظ الفرق ، هنا يجب أن نحدد ارتفاع وعرض الشاشة لأن السطح الخلفي سطح غير مرئي وبذلك ليس شرطا أن يكون ارتفاعه وعرضه بعرض الشاشة وارتفاعها كما كان الحال مع السطح الرئيسي. لذلك سوف نستخدم الأعضاء

dwWidth و dw.Hegiht لإعطاء القيم المناسبة (في هذا المثال سوف نجعل السطح الخلفي مساويا مساحة الشاشة).

```
ddrval = DD4->CreateSurface (&ddsd, &lpDDS4Back , NULL);
if ( ddrval != DD_OK) return FALSE;
```

المؤشر إلى السطح الخلفي والذي عرفناه في أول الملف.

طبعا نقوم بالتأكد أن العملية مرت بنجاح عن طريق اختبار قيمة المتغير " ddrval " وهل هي OK أم لا.

والخطوة الأخيرة هي إلحاق (أو لصق) السطح الخلفي بالسطح الرئيسي كما يلي :

```
ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
ddrval=lpDDS4Primary->GetAttachedSurface(&ddscaps, &lpDDS4Back);
```

لاحظ أننا استخدمنا المتغير ddscaps والذي عرفناه وناقشناه في بداية هذه الوظيفة وأوضحنا أنه كان باستطاعتنا استخدام العضو "ddsd.ddsCaps" أحد أعضاء المؤشر "ddsd" ولو فعلنا ذلك ستكون الخطوة الأخيرة كما يلي :

```
ddsd.ddsCaps.dwCaps = DDSCAPS_BACKBUFFER;
ddrval=lpDDS4Primary->GetAttachedSurface(&ddsd.ddsCaps,
&lpDDS4Back);
```

الفصل الثالث عشر

البنية الداخلية لـ

AGDX

البنية الداخلية لـ AGDX

قبل أن نتعمق في هذا الموضوع أحب أن انوه الأخ و الأخت القارئة بأن معرفة طريقة برمجة الكائنات في لغة السي المحسنة مهمة لفهم بقية مواضيع الكتاب (في الحقيقة ليس هناك الكثير لتعلمه عن الكائنات في لغة السي المحسنة ، ولكن مراجعتها سوف يفسر لك الكثير من التساؤلات) تقريبا جميع كتب لغة السي المحسنة تتكلم عن الكائنات (أو يطلق عليها في لغة السي المحسنة Class Objects) ولكن سوف نقوم بشرحها بشكل مبسط وسريع كما يلي :

الكائنات (CLASS) في برمجة Windows9X ولطبيعة البرمجة على هذه الأنظمة ذات القدرة على تشغيل أكثر من برنامج في وقت واحد (multithreaded operating system) فإننا سوف نشرحها بشكل مبسط وسريع الكائن (Class Object) هو صورته مطوره عما يعرف في لغة السي (C) بـ (Structure) وهي عبارة عن مجموعة متغيرات من أنواع مختلفة (قد تكون من نوع int أو float وهكذا , راجعها في الفصل الأول من هذا الكتاب لمعلومات أكثر). وتعتمد لغات البرمجة الكائنية (مثل لغة السي المحسنة C++) ومكتبة الألعاب على الكائنات في تعاملها مع البرامج. وتكمن أهمية الكائنات في أنها تجعل من برنامجنا أكثر تنظيماً. فإذا حدث خطأ ما لسبب معين فإننا نقوم بالتأكد من الكائن المسؤول عن ذلك الخطأ بدون الحاجة للبحث في جميع الشفرة البرمجية. هذا بالإضافة إلى أسباب عديدة أخرى جعلت من البرمجة الكائنية هي الطريقة المثلى في التعامل مع شفرات البرامج اليوم. ويكون شكل كائن مبسط كما يظهر في الصفحة التالية :

Class أسم الكائن

{
public:

أسم الكائن {

:

يبدأ الكمبيوتر في النظر هنا عند بداية الكائن

ويسمى هذا المكان الباني (constructor)

وهو مكان جيد لحجز الذاكرة باستخدام الأمر

new

:

}

{ أسم الكائن الوارث~

:

عند الانتهاء من الكائن فإن الكمبيوتر ينظر هنا لذلك هذا

هو المكان الجيد لتحرير الذاكرة باستخدام (Delete) إذا

كنا قد استخدمنا new في الباني (Distractor) ويتميز

بهذه العلامة ~ قبل اسم الكائن كما نرى في هذا المثال

:

}

private:

.

}

Surface Class

من الآن سوف نفترض معرفة القارئ بالأساسيات لبرمجة الكائنات في لغة السي المحسنة ونعود إلى الكائنين الذين سبق وتحديثنا عنهما (Surface و Screen) نقوم أولاً بالنظر إلى كائن الـ Surface كما يلي :

```
AGDXSurface::AGDXSurface(AGDXScreen *pScreen, int Width, int Height)
{
    Create(pScreen, Width, Height);
}
```

هذا هو الباني وكما نرى عندما نريد استخدام هذا العنصر فإننا نقوم بتمرير كائن الشاشة "Screen" (سوف نتكلم عنة بعد قليل) بالإضافة إلى العرض ولارتفاع المطلوب للسطح ، فيقوم الباني بدورة بتمرير هذه القيم إلى أحد الأعضاء (لكل

```
BOOL AGDXSurface::Create(AGDXScreen *pScreen, int Width, int Height)
{
    HRESULT rval;

    ZeroMemory(&m_DDS, sizeof(m_DDS));
    m_DDS.dwSize = sizeof(m_DDS);
    m_DDS.dwFlags = DDS_CAPS | DDS_HEIGHT | DDS_WIDTH;
    m_DDS.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN;
    m_DDS.dwWidth = Width;
    m_DDS.dwHeight = Height;

    rval = pScreen->m_lpDD->CreateSurface(&m_DDS, &m_lpDDS, NULL);
    if(rval != DD_OK)
    {
        DDError(rval, pScreen->m_hWnd);
        return FALSE;
    }

    return TRUE;
}
```

كائن أعضاء يطلق عليهم **Member Functions** ، مرة أخرى ، راجع الكائنات في أي كتاب للغة السي لمعلومات إضافية عنها) وهو العضو **Create**. دعونا الآن ننظر إلى العضو **Create** في المربع بالصفحة السابقة :

أولا نقوم بالتأكد أن الذاكرة المطلوبة خالية من أي معلومات سابقة ، ونقوم بهذه العملية عن طريق استخدام أحد أوامر **win32** وهو الأمر **ZeroMemory()** حيث يقوم بوضع قيم خالية (صفر يعتبر قيمة خالية) في مكان الذاكرة المطلوبة وبذلك نتأكد بأن المكان المطلوب في الذاكرة لا يحتوي على شيء. ويستخدم هذا الأمر عنوان المكان المطلوب في الذاكرة (في هذه الحالة استخدمنا المؤشر **m_DDSD**) وكذلك الحجم المطلوب بالبايت (استخدمنا الأمر **sizeof()** لمعرفة حجم المكان المطلوب عن طريق المؤشر **m_DDSD**):

```
ZeroMemory(&m_DDSD, sizeof(m_DDSD))
```

كذلك نرى أن هذا العضو يأخذ هذه القيم الممررة إليه من الباني ويقوم بوضعها في المتغير **m_DDSD** (وهو المتغير "أو المؤشر" المسؤول عن حجم السطح المطلوب)، بعد ذلك يستخدم كائن الشاشة "Screen Class" (والذي سبق ومررناه في الباني، نحن نعلم انه عند استخدام أي ممثل "instance" لكائن معين، كما هو الحال هنا، فإننا نستطيع استخدام جميع أعضاء ذلك الكائن عن طريق هذا الممثل) وعن طريقة يمكننا إنشاء السطح كما يلي:

```
rval = pScreen->m_lpDD->CreateSurface(&m_DDSD, &m_lpDDS, NULL);
```

حيث استخدمنا المتغير `m_lpDD` (هذا كائن ممثل لـ `DirectDraw` سبق وأنشأناه في الكائن `Screen` سوف نتكلم عنة اكثر عندما نشرح ذلك الكائن) ومن خلاله استخدمنا أحد أوامر `DirectDraw` وهو الأمر `CreateSurface()` لإنشاء السطح المطلوب. أما المتغير `m_lpDDS` فهو مجرد مؤشر لهذا السطح (عندما ترى العلامة "&" أمام متغير معين فاعلم أن هذا المتغير هو مؤشر لشيء معين).

ثم أخيرا نرى الأمر `DDError()` وهذا الأمر مهم جدا عند إنشاء برامجنا فعن طريقة نستطيع أن نعرف السبب إذا لم يعمل البرنامج. كما سبق و تعلمنا في المثال السابق فإننا قمنا بإنشاء كائن يقوم بحفظ حالة كل أمر من أوامر `DirectX` (هنا استخدمنا متغير أسميناه `rval`) بحيث لو حصل أي خطأ يقوم الأمر `DDError()` بمقارنة قيمة المتغير `rval` مع الأمر الذي سبب ذلك الخطأ فيعرف ما هو الخطأ ويقوم بإعطائنا ذلك السبب على شكل رسالة على الشاشة:

```
if(rval != DD_OK)
{
    DDError(rval, pScreen->m_hWnd);
    return FALSE;
}
```

ما هو الجديد في الكائن `Surface` ؟



مع العلم أننا في هذا الكائن نتصور بأننا أنشأنا شي جديد ولكن في الحقيقة ماعدا الأمر `ZeroMemory()` واستخدام الأمر `DDError()` ليس هناك جديد (حتى هذين الأمرين وضعنا فقط للتأكد من أن كل شيء على ما يرام ويمكن حذفهما لو أردنا

وسيقوم البرنامج بعمله في كل الأحوال) فكل ما قمنا به سبق وشرحناه في المثال " win32+Draw " وكل ما فعلناه هنا أننا قمنا بترتيب نفس الخطوات لإنشاء سطح على الذاكرة في كائن خاص به.

Screen Class

بعد أن تحدثنا عن الكائن الأول وهو الكائن Surface نتحدث الآن عن الكائن الآخر وهو الكائن Screen . أولاً يجب أن اذكر القارئ العزيز أننا لن نقوم بإنشاء أي كائنات أخرى للعنصر DirectDraw ما عدا هذين الكائنين ولو أردنا إضافة أي مميزات لـ DirectDraw لاستخدامها في برامجنا سوف نقوم بتلك الإضافات في هذين الكائنين. نحن في الكائن الأول قمنا بإنشاء أسطح في ذاكرة الكمبيوتر ، هذا كل ما عملناه هنا ، ولكي نستفيد من هذه الأسطح كان لابد من إنشاء كائن يقوم باستغلالها لإظهار الحركة على الشاشة لذلك نرى الحاجة الماسة لإنشاء الكائن Screen.

أولاً سوف ننظر إلى الباني :

```
AGDXScreen::AGDXScreen()
{
    m_lpDD = NULL;
    m_lpDDFront = NULL;
    m_lpDDBack = NULL;
}
```

كما نرى أن الباني لهذا الكائن بسيط الفهم ، فكل ما يقوم به هو عطاء قيمة خالية "NULL" لكل من :

المتغير m_lpDD (هذا متغير "أو مؤشر" ممثل لـ DirectDraw أنشأناه في هذا الكائن وقد سبق واستخدمناه في الكائن السابق "Surface")

والمتغير `m_lpDDSDFront` (وهو المتغير "أو المؤشر" المسؤول عن السطح الرئيسي) والمتغير `m_lpDDSDBack` (وهو المتغير "أو المؤشر" المسؤول عن السطح الخلفي).

ماذا تقصد بقيمة خالية "NULL" ؟



بما أن جميع المتغيرات الموجودة في الباني هي عبارة عن مؤشرات (Pointers) تقوم بدورها في حمل قيمة عنوان معين في الذاكرة فإن إعطاء القيمة "NULL" يعبر عن أن المؤشر لا يؤشر إلى أي شيء في تلك اللحظة بمعنى انه لا يحمل أي عنوان.

بعد أن رأينا الباني الأول دعونا نرى الباني الثاني للكائن `Screen` (نعم يمكننا أن نستخدم بانينين لأي كائن في لغة السي المحسنة، وكما نعلم بأن الباني سواء الأول أو الثاني هو عبارة عن عضو من الأعضاء ولكن يقوم الكائن بالنظر إليه قبل الأعضاء الأخرى)

ونرى الباني الثاني كما يلي :

```
AGDXScreen::AGDXScreen(void *hWnd, DWORD Width, DWORD Height,
DWORD BPP, BOOL bVGA)
{
    CreateFullScreen(hWnd, Width, Height, BPP, bVGA);
}
```

كما نرى بأننا عند استخدام الكائن `Screen` في برنامجنا نقوم بتمرير رقم النافذة `hWnd` ثم عرض الشاشة `Width` وارتفاعها `Height` ، و الألوان المستخدمة

بالبت لكل نقطة على الشاشة "BPP" وأخيرا الاختياريين صحيح "True" أو غير صحيح "False" للمتغير bVGA (هذا المتغير يقوم بتحويل الشاشة إلى البعد..... في حالة لو كان صحيح)

فيقوم الباني بدورة بتمرير هذا المعطيات إلى العضو CreateFullScreen() وهو العضو المسؤول عن إنشاء واستغلال الشاشة بكاملها.

لقد قمنا في هذا الكائن بإنشاء ثلاث أعضاء فقط لحاجتنا لها (سوف نقوم بشرحها بالتفصيل بعد قليل) وهي العضو الأول المسؤول عن استغلال الشاشة بكاملها CreateFullScreen() ثم العضو الثاني وهو عضو التعبئة بلون معين Fill() ويقوم بتعبئة السطح الخلفي بلون معين وأخيرا عضو القلب Flip() وهو المسؤول عن قلب الشاشة الخلفية إلى الشاشة الرئيسية.

وذلك لأن كل ما نريده في هذه النقطة هو تحويل مثالنا الأول "win32+Draw" لاستخدام الكائنات بحيث نعبئ السطح الخلفي بلون معين ثم قلب ذلك السطح إلى السطح الرئيسي لرؤيته. ونستطيع طبعا وبنفس الأسلوب أن نضيف أعضاء أخرى كلما احتجنا لإضافة ميزة معينة من مميزات DirectDraw.

• العضو CreateFullScreen() :

قبل أن نرى مما يتكون هذا العضو أرجو من القارئ الكريم مراجعة أول مثال سبق وشرحناه "win32+Draw" وبالذات مراجعة الجزء DirectDrawInit() والتأكد من فهمه. سوف تلاحظ الاختلاف البسيط بينه وبين هذا العضو وسبب الاختلاف أننا في هذا العضو نملك أولاً كائن خاص بالسطح (وهو الكائن Surface)، سواء كان السطح الخلفي أو الأمامي أو أي سطح إضافي. ثانياً في ذلك المثال استخدمنا الإصدار الأول للعضو DirectDraw وهو الإصدار المتضمن في DirectX1 وبما أننا في هذا الكتاب نستخدم آخر إصدار من إصدارات DirectX سوف نستخدم الإصدار الثاني للعضو DirectDraw ولكي تكون الأمور سهلة ولعدم الحاجة إلى طريقة جديدة لاستخدام هذا الإصدار الجديد فإن شركة Microsoft قامت بإضافة أمر اسمه QueryInterface يقوم بتحويل أي إصدار قديم لاستغلال الإصدار الجديد (نعم وبهذه السهولة) ونرى هذا العضو كما يلي :

```

BOOL AGDXScreen::CreateFullScreen(void *hWnd, DWORD Width,
DWORD Height, DWORD BPP, BOOL bVGA)
{
    HRESULT rval;
    DWORD dwFlags;
    DDSURFACEDESC ddsd;
    DDSCAPS ddscaps;

    m_lpDD = NULL;
    m_bFullScreen = TRUE;
    m_dwPixelWidth = Width;
    m_dwPixelHeight = Height;
    m_BPP = BPP;
    m_hWnd = hWnd;

    dwFlags = DDSCL_EXCLUSIVE | DDSCL_FULLSCREEN |
              DDSCL_ALLOWREBOOT | DDSCL_ALLOWMODEX;

    LPDIRECTDRAW lpDD;

```

أولاً قمنا بإعطاء قيم مبدئية لجميع المتغيرات المستخدمة في هذا العضو (جميع التعاريف لجميع المتغيرات المستخدمة في كل الكائنات لهذه المكتبة موجودة في ملف واحد يمكنك الإطلاع عليه وهو الملف "AGDX.h").

الآن لنلاحظ كيف استطعنا استخدام الإصدار الثاني لـ DirectDraw عن طريق الأمر : QueryInterface

```
rval = DirectDrawCreate(NULL, &lpDD, NULL);
if(rval != DD_OK) DDError(rval, hWnd);

rval = lpDD->QueryInterface(IID_IDirectDraw2,
                           (LPVOID*)&m_lpDD);
if(rval != DD_OK) DDError(rval, hWnd);

RELEASE(lpDD);
```

بعد ذلك نرى في الجزء الباقي من هذا العضو أننا قمنا بنفس العملية التي سبق وشرحناها في المثال الأول لطريقة استغلال الشاشة بكاملها:

```
rval = m_lpDD->SetCooperativeLevel(hWnd, dwFlags);
if(rval != DD_OK) DDError(rval, hWnd);

if(bVGA) rval = m_lpDD->SetDisplayMode(m_dwPixelWidth,
m_dwPixelHeight, m_BPP, 0, DDSDM_STANDARDVGMODE);

else rval = m_lpDD->SetDisplayMode(m_dwPixelWidth,
m_dwPixelHeight, m_BPP, 0, 0);
if(rval != DD_OK) DDError(rval, hWnd);

ddsd.dwSize = sizeof(ddsd);
ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;

ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE |
DDSCAPS_FLIP | DDSCAPS_COMPLEX;

ddsd.dwBackBufferCount = 1;

m_lpDDFront = new AGDXSurface;
m_lpDDSBack = new AGDXSurface;
```

```

        rval = m_lpDD->CreateSurface(&ddsd, &m_lpDDFront-
>m_lpDDS,
                                                    NULL);
        if(rval != DD_OK) DDError(rval, hWnd);

        ddscaps.dwCaps = DDSCAPS_BACKBUFFER;

        rval = m_lpDDFront->m_lpDDS->GetAttachedSurface(&ddscaps,
&m_lpDDSBack->m_lpDDS);
        if(rval != DD_OK) DDError(rval, hWnd);

        return TRUE;
    }

```

ولقصد التشجيع احب أن أنبه على أن هذا العضو هو أهم عضو نستخدمه ليس فقط في هذه المرحلة من الشرح بل في الكتاب ككل. لذلك عزيزي القارئ حاول أن تدرسه بتأني وراجعة اكثر من مرة حتى تستطيع أن تبدأ في فهم كيفية عمل تكنولوجيا DirectX بشكل مريح وسوف يساعدك كثيرا فهم المثال "win32+Draw" وخاصة طريقة إنشاء كائن DirectDraw في الجزء DirectDrawInit() من ذلك المثال.

• العضو Fill() :

كما سبق وأشرنا أننا نريد أن نجعل هذا المثال يقوم بنفس عمل المثال السابق "win32+Draw" مع اختلاف بسيط وهو استخدام الكائنات ، وكان السبب هو أن الكائنات سوف تعطينا شيء من النظام والسهولة مع تزايد حجم برامجنا لذلك نحتاج إلى عضو يقوم بتعبئة الشاشة بلون معين كما هو الحال هنا :

```

void AGDXScreen::Fill(DWORD FillColor)
{
    DDBLTFX ddBlTFx;

    ddBlTFx.dwSize = sizeof(DDBLTFX);

```

```

ddBlitFx.dwFillColor = FillColor;
m_lpDDSBack->m_lpDDS->Blit(NULL, NULL, NULL,
DDBLT_WAIT | DDBLT_COLORFILL, &ddBlitFx);
}

```

في البداية يقوم هذا العضو بأخذ رقم اللون المطلوب تعبئته ويخزنه في المتغير "FillColor". بعد ذلك ننشئ الممثل "ddBlitFx" للكائن "DDBLTFX" ولكن لماذا؟ لأنه كما سبق وذكرنا بأننا سنعمل على إضافة كل المميزات التي نريد إظهارها على الشاشة في هذا الكائن (الكائن Screen()) وما نعمله أولاً هو النظر إلى إمكانيات تكنولوجيا DirectX. فإذا كانت تلك الميزة المطلوبة (تعبئة الشاشة بلون معين في هذه الحالة) موجودة، قمنا باستخدامها وإلا فإننا نقوم ببرمجة تلك الإمكانية بأنفسنا. لحسن الحظ أن تكنولوجيا DirectX تملك الكثير من الإمكانيات كما هو الحال هنا فعن طريق استخدام الكائن "DDBLTFX" (وهو الكائن المهتم بالمؤثرات، وهو جزء من العنصر DirectDraw) نستطيع أن نقوم بمليء الشاشة بأي لون نشاء (هناك إمكانيات أخرى لهذا الكائن، سوف نذكرها كلما احتجنا لذلك). بعدما أنشأنا الممثل قمنا عن طريقة بإعطاء عناصر الكائن "DDBLTFX" القيم المطلوبة، مثل المربع الذي يجب ملئته (في هذه الحالة المربع يشمل الشاشة بأكملها) وكذلك اللون المطلوب (اللون موجود في المتغير "FillColor") وأخيراً قمنا باستخدام الأمر "Blit" أحد أوامر العنصر DirectDraw ليقوم بوضع ذلك المربع ذو اللون المطلوب على السطح الخلفي.

(يجب استخدام الأمر "Blt" وليس الأمر "BltFast" ، وهذين الأمرين يقومان بنفس العملية وهي وضع صورة معينة على السطح المطلوب وفي حالة عدم وجود أي سرعات على الجهاز المستخدم فإن الأخير أسرع بنحو 10٪ ولكنه محدود الإمكانيات كما في حالتنا هذه)



فرضاً أنت لم تقم بشرح هذه العملية كيف أستطيع أن أعرف أو أتذكر استخدام كل هذه الأوامر وبنفس الأسلوب للوصول إلى الهدف المطلوب وهو تعبئة الشاشة بلون معين في هذه الحالة؟؟



ليس الهدف من البرمجة هو الحفظ ، ولكن الهدف هو الفهم فكل ما عليك فعله عندما تريد أن تقوم بعمل معين هو النظر إلى الأمثلة الموجودة ، سواء التي تأتي مع تكنولوجيا DirectX أو مع هذا الكتاب (واعتقد أن هذا الكتاب أشتمل على الكثير من الأمثلة والتي تجيب على غالبية التساؤلات).

• العضو Flip() :

وأخيرا عضو القلب Flip() وهو المسؤول عن قلب السطح الخلفي إلى السطح الرئيسي وعن طريقة تقوم بأمر الكمبيوتر بالقيام بهذه العملية في أي مرحلة نشاء في برنامجنا (عادة نقوم بذلك عندما نقوم بتغيير معين في سطح غير السطح الرئيسي ونريد إظهار ذلك التغيير) أولا دعونا نرى هذا العضو :

```
HRESULT AGDXScreen::Flip(void)
{
    HRESULT rval;

    rval = m_lpDDSDraw->m_lpDDS->Flip(NULL, DDFLIP_WAIT);

    return rval;
}
```

كما نلاحظ أن كل ما يقوم به هذا العضو هو استخدام الأمر "Flip" أحد أوامر العنصر DirectDraw ويقوم بإرجاع حالة تلك العملية ، إذا كانت ناجحة أم لا عن طريق استخدام المتغير "rval" كما هو الحال مع جميع الأعضاء.

البرنامج

الآن الأمور أصبحت سهلة و مسلية ، ليس هناك الكثير لتعلمه في البرنامج الرئيسي. واصبح لدينا كائنين هنا، كائن السطح "Surface" وكائن الشاشة "Screen" وكل ما علينا فعله هو استخدامهما في برنامجنا.

برنامجنا فهو عبارة عن برنامج Win32 كما سبق وشرحناه مع بعض الإضافات لاستغلال الكائنين السابقين بحيث نستخدم عن طريقهما العنصر DirectDraw هذا طبعا سوف يجعل من برنامجنا الرئيسي سهل الفهم وأبسط بكثير مما كان عليه من قبل.

بعد كتابة برنامج Win32 نقوم أولا بتعريف كائن الشاشة كما يلي :

```
AGDXScreen* Screen;
```

وكل برامجنا التي تستخدم هذه المكتبة سوف تقوم أولا بتعريف وإنشاء هذا الكائن. الخطوة الثانية هي إنشاء الكائن بعدما عرفناه وعن طريقة نقوم بتحديد عرض وارتفاع الشاشة وكذلك عدد الألوان (عن طريق وضع عدد البت لكل نقطة على الشاشة) ونرى هذه العملية كما يلي :

```
Screen = new AGDXScreen();
Screen->CreateFullScreen(hWnd, 640, 480, 8);
```

الآن اصبح برنامجنا يستخدم العنصر DirectDraw بهذه السهولة (لاحظ أيها القارئ العزيز أن وجود الكائنين (Surface و Screen) وكذلك DXError من ضمن برامج هذا المشروع هو لقصد التوضيح لأننا في الحالة العادية تكون كل الكائنات

المستخدمة موجودة من ضمن المكتبة ولا يوجد داعي لإلحاقها بالمشروع، كما هو الحال مع أمثلة الجزء الأول من هذا الكتاب (لتتذكر أننا في أول مثال شرحناه في هذا الفصل عن DirectDraw (وهو المثال Win32+Draw) قمنا بتدوير الألوان على الشاشة ، نحن هنا سوف نقوم بعمل مشابه بجعل الشاشة تتغير بين اللون الأحمر و الأزرق. كيف يتم ذلك ؟ أولاً يقوم البرنامج بإنشاء سطحين (عن طريق الكائن Surface) ثم يقوم بتعبئة السطح الخلفي (عن طريق الأمر Fill) أحد أعضاء الكائن Screen) باللون الأحمر ثم يقلبه للسطح الرئيسي (عن طريق استخدام العضو Flip() أحد أعضاء الكائن Screen). في نفس الوقت يقوم بتعبئة السطح الخلفي مرة أخرى وبنفس الطريقة باللون الأزرق ثم يقلبه إلى السطح الرئيسي و هلم جراً. ونرى هذه العملية كالتالي :

```

case WM_TIMER:
{
    if(Toggle)
    {
        Screen->Fill(4); // Fill the back buffer red
        Toggle = 0;
    }
    else
    {
        Screen->Fill(1); // Fill the back buffer blue
        Toggle = 1;
    }
    Screen->Flip(); // Flip the back buffer to the front
}
break;

```

هذا كل البرنامج ، نلاحظ القدر الكبير لذي اختصرناه في البرمجة نتيجة استخدام الكائنات. ولكن يجب أن نقوم بإضافة مميزات جديدة للكائنين السابقين حتى يكونا جاهزان للاستخدام بشكل فعال ، لذلك سوف نقوم بإضافة عدة أعضاء للكائنين حتى نتمكن من استغلال كل إمكانيات العنصر `DirectDraw` عن طريقهما.

ملفات الصورة مستقلة النوعية Device-Independent Bitmap

نظام ويندوز وبتالي DirectX تستخدم نظام ملفات يطلق عليه "ملف صورة مستقل النوعية" أو Device-Independent Bitmap (DIB) كنظامها المفضل للتعامل مع الصور ويتكون ملف DIB من معلومات عن أبعاد الصورة المحفوظة فيه وعدد ألوانها والقيم التي تمثل تلك الألوان ومعلومات عن كل نقطة في تلك الصورة بالإضافة إلى أن DIB يحتوي على بعض المعلومات المهمة للمستعمل كمعلومات عن الضغط (تستخدم طريقة ضغط المعلومات لتوفير جزء من الذاكرة المستخدمة) والألوان المستخدمة (إذا لم تكن كل الألوان مستخدمة) و حجم الصورة (في حالة لو أردنا طباعتها) وملفات الـ DIB عادة تحمل الإضافة "bmp" وكذلك "dib". ولأن هذا النوعية من الصور تستخدم كثيرا في برمجة ويندوز فإن الأوامر المتوفرة تحت هذا النظام كثيرة ونستطيع استخدامها مع DirectX . مثلاً لننظر إلى الوظيفة (Function) التالية والمأخوذة من الملف dduil.cpp الذي يأتي من ضمن المجموعة البرمجية لـ DirectX ويقوم بتعبئة صورة DIB إلى DirectX surface :

```
extern "C" IDirectDrawSurface * DDLoadBitmap (IdirectDraw *pdd,
      LPCSTR szBitmap, int dx, int dy)
{
  HBITMAP      hbm;
  BITMAP       bm;
  DDSURFACEDESC ddsd;
  IDirectDrawSurface *pdds;

  //
  // This is the Win32 part.
  // Try to load the bitmap as a resource, if that fails, try it as a file.
  // هذا الجزء مأخوذ من برمجة نظام التشغيل Win32
  // حاول تعبئة هذه الصورة عن طريق الموارد فإذا لم يكن ذلك ممكن حاول عن طريق تعبئة الملف//

  hbm = (HBITMAP)LoadImage(GetModuleHandle(NULL), szBitmap,
    IMAGE_BITMAP, dx, dy, LR_CREATEDIBSECTION);
}
```

```
if (hbm == NULL)
    hbm = (HBITMAP)LoadImage(NULL, szBitmap, IMAGE_BITMAP, dx, dy,
LR_LOADFROMFILE|LR_CREATEDIBSECTION);

if (hbm == NULL)
    return NULL;

//
// Get the size of the bitmap.
// أحصل على حجم الصورة
GetObject(hbm, sizeof(bm), &bm);

// بعد ذلك أرجع إلى البرنامج الرئيسي
// Now, return to DirectX function calls.
// Create a DirectDrawSurface for this bitmap.
//
ZeroMemory(&ddsd, sizeof(ddsd));
ddsd.dwSize = sizeof(ddsd);

ddsd.dwFlags = DDSD_CAPS | DDSD_HEIGHT | DDSD_WIDTH;
ddsd.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN;
ddsd.dwWidth = bm.bmWidth;
ddsd.dwHeight = bm.bmHeight;

if (pdd->CreateSurface(&ddsd, &pdds, NULL) != DD_OK)
    return NULL;

DDCopyBitmap(pdds, hbm, 0, 0, 0, 0);

DeleteObject(hbm);

return pdds;
}
```


الفصل الرابع عشر



واجهة تكنولوجيا Direct 3D

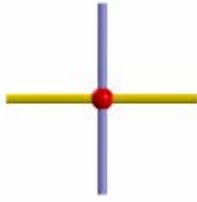
العنصر المسؤول عن البعد الثالث في تكنولوجيا

DirectX

في هذا الفصل سوف نتعرف على جميع أوامر هذه التكنولوجيا وشرح كل أمر على حدا. وفي الفصل القادم سوف نقوم باستخدام هذه الأوامر لإنشاء مثال ثلاثي الأبعاد.

البعد الثالث

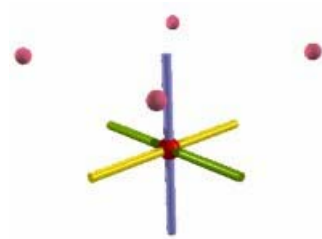
في الفضاء يوجد أكثر من بعد ، ولكن على شاشة الكمبيوتر يوجد بعدين فقط هما البعد السيني و البعد الصادي (س،ص) :



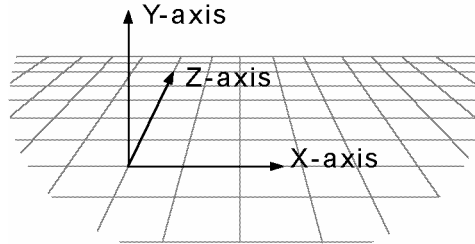
ولو أردنا أن نحدد نقطة ما على هذين البعدين فإننا سوف نقوم بإعطائها إحداثيتين سيني و صادي :



ولأن كل الأجسام في الفضاء لها أكثر من بعدين وحتى نكون أقرب إلى الواقع فإننا نحتاج إلى البعد الثالث :



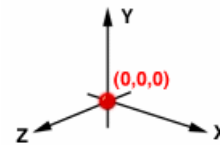
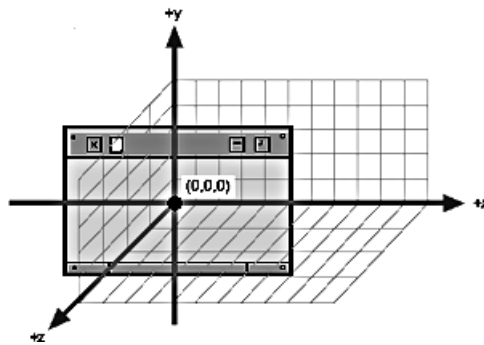
فالبعد السيني هو العرض والبعد الصادي هو الارتفاع ولكن البعد الثالث هو العمق
 فلذلك نحتاج إلى ثلاث أرقام حتى نحدد أي نقطة في الفضاء (Z,Y,X)
 واضح في الرسم التالي :



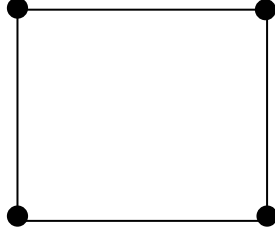
لماذا البعد الثالث يشكل أي أهمية؟؟ البعد الثالث هو الطريقة المثلى لإنتاج برامج
 تمثل الكون المحيط بنا لأننا نضع في الحسبان الأجسام كما هي في الواقع ، كل جسم
 في الواقع مثل الكتاب له طول وعرض وارتفاع ولكي نمثل هذا الكتاب على شاشة
 الكمبيوتر بشكل واقعي فإننا نحتاج الأبعاد الثلاثة لهذا الجسم.

نقطة الصفر أو نقطة الأصل :

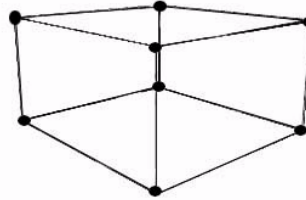
في المحيط الثلاثي الأبعاد x,y,z لابد وأن تكون هناك وحدة قياس بحيث
 نستطيع أن نصف مكان أي نقطة في الفضاء وتبدأ وحدة القياس من نقطة الأصل
 (0,0,0) كما هو واضح من الشكل التالي :



وبذلك نستطيع وصف أي نقطة في الفضاء أو أكثر. والأجسام في الواقع هي عبارة عن مجموعة نقاط مختلفة متصلة ببعضها البعض. مثلا نستطيع أن نعبر عن مربع عن طريق أربع نقاط مختلفة لكل زاوية من المربع كما في الشكل التالي:

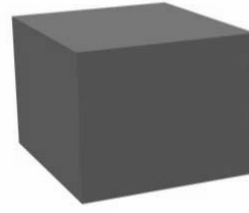


ولاحظ أن شكل المربع هو عبارة عن خطوط توصل تلك النقاط ببعضها البعض، ولو أخذنا شكل آخر مثل الصندوق بحيث نري النقاط موصلة كما في الشكل التالي: -

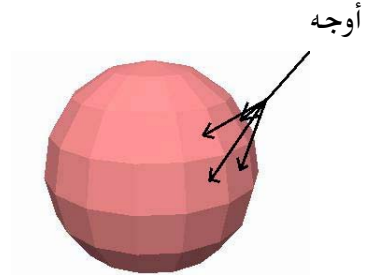


وتسمى طريقة العرض هذه **wireframe** أو "الإطار السلكي" وهي أسرع طرق العرض لدى الكمبيوتر فكل ما على الكمبيوتر فعله هو أن يوصل النقاط ببعضها البعض لتتكون صور الأجسام. وكانت هذه هي الطريقة الأساسية في عرض الأبعاد الثلاثية على شاشة الكمبيوتر لمدة طويلة.

ولكن هذه الطريقة ظلت بعيدة عن الواقع ، مثلا الصندوق ليس مجرد نقاط فقط بل أن هناك ألوان مختلفة لكل وجه من أوجهه بحيث تعطي الشكل العام للصندوق :



لاحظ بأن للصندوق ستة أوجه تمثل جسم الصندوق وقد يكون هناك أكثر من ستة أوجه لجسم معين بحيث تعطي شكل مختلف كما في المثال التالي نرى كرة مكونة من أكثر من وجه يمثلن في مجموعهن شكل الكرة ولكل وجه عدة نقاط تسمى Vertices قد يزيد أو يقل عددها لكل وجه متصلة بنقاط أخرى لوجه آخر وهكذا (في حالة الصندوق هنا فإن كل وجه يتكون من أربع نقاط). وكلما زاد عدد الأوجه لجسم معين كلما زاد قربها من الشكل الواقعي ولكن زيادة عدد الأوجه يؤدي إلى زيادة عدد النقاط لذلك الجسم مما سوف يضيف الكثير من العمليات الحسابية التي سوف يقوم بها الكمبيوتر لكي يتحكم في ذلك الشكل.



وإضافة الألوان لكل وجه من الأوجه هي الطريقة الأخرى لإعطاء الواقعية على شكل الجسم المعروض وتسمى تلك الطريقة **Flat** أو السطحية وهي أكثر واقعية من الطريقة الأولى ولكنها ليست الطريقة المثلى لأن شكل الجسم يظهر على أنه مجموعة من الأوجه.

مصدر الإضاءة

أن عين الإنسان تلاحظ اختلاف الإضاءة حولها، فعندما ننظر إلى لون معين مثل كتاب احمر اللون تحت ضوء المصباح ثم نطفي ذلك المصباح فإن اللون الأحمر سوف يتغير إلى أحمر أكثر غمقا وذلك لأن المصباح أعطى انعكاس اكبر للون الأحمر فنراه مختلفا عنة بدون تلك الإضاءة. لذلك عند أخذ مصدر الضوء في الحسبان فإننا نحصل على نتائج قريبة من الواقع. ويمكننا تحديد مصدر الضوء من خلال الانعكاسات المتولدة عنة ، مثلا تكون الألوان فاتحة عندما تقترب من مصدر الضوء و تميل إلى الأغمق عند الابتعاد عنة.

عندما نريد أن نضع عالم واقعي على الكمبيوتر فإننا لا نرى بدا من استخدام الأبعاد الثلاثية كما هو معروف في "التصور الحقيقي" حيث نقوم بإدخال معلومات كاملة عن عالم معين فيقوم الكمبيوتر ببناء ذلك العالم وبذلك نستطيع أن نقوم باختبارات عدة على الكمبيوتر بدون وجودنا هناك وبالتالي لسنا معرضين لأي خطر حقيقي. مثلا يتم تدريب الطيارين على أجهزة الكمبيوتر بدون تعريضهم أو تعريض الطائرات المكلفة الثمن لأي خطر من خلال هذه التكنولوجيا. فيستطيع الطيار أن يقوم برحلة جوية ورحلات قتالية بدون أي خطر ، وكل ما في الأمر أن الكمبيوتر يقوم بإعطائنا النتيجة في حالة فشل المهمة أو نجاحها. كذلك تستخدم هذه التكنولوجيا في الألعاب ، فنستطيع أن نذهب من خلال الكمبيوتر إلى عوالم أخرى كانت مجرد خيالات فقط.

Direct-3D

سنعمل في هذا الفصل على التعرف على هذه التكنولوجيا و كل الأوامر التي توفرها بدون التطرق لبرامج معينة و تفاصيلها البرمجية ، بمعنى أننا لن نتطرق إلى البرمجة حتى نشعر بارتياح مع كل الأوامر المتوفرة لدينا. سوف نرى كيفية استخدام كل أمر وما هي فائدته بحيث نستطيع أن نرى صورة واضحة لهذه التكنولوجيا.

Direct-3D هي أحد عناصر تكنولوجيا دايركت أكس (DirectX). والجميل في هذه التكنولوجيا هو التعامل المباشر مع جهاز الكمبيوتر، في السابق كان لمن يريد البرمجة في نظام Windows عالية أن يبرمج عن طريق الطبقة العليا من هذا النظام. حيث يقوم البرنامج بطلب أي تغيير في الصوت أو الصورة من نظام التشغيل ثم إن نظام التشغيل بدوره يقوم بالتعامل مع الجهاز مما ينتج عنه تأخير ملحوظ في تلك العملية فأدى ذلك إلى جعل بيئة Windows مناسبة للبرامج الاقتصادية أو المصرفية التي لا تتطلب سرعة في تغيير الألوان على الشاشة و لا تحتاج إلى الأبعاد الثلاثية في عملها ويكون تركيزها فقط على أساس المعلومات المخزنة و سرعة جلبها من المصدر الموجودة فيه.

وكان ولا زال نظام windows بيئة مثالية لتلك النوعية من البرامج حيث تضي عليها سهولة التعامل و حسن واجهة المستخدم. أما البرامج التي تستخدم الأبعاد الثلاثية فإنها تتطلب الكثير من السرعة ، ولذلك في البداية لم يكن نظام windows البيئية المثالية لهذه البرامج لأنها تحتاج إلى تعامل مباشر مع الكمبيوتر دون أن يكون هناك وسيط مثل نظام التشغيل. من هذا المنطلق ولدت فكرة Direct3D التي أعطت القدرة على التعامل بشكل مباشر مع الجهاز. ولم تتوقف هذه التكنولوجيا على ذلك بل أضافت القدرة على استغلال المسرعات الموجودة واستغلالها بدون الاهتمام إلى اختلاف

أنواعها لأنها جميعاً مصممة لاستغلال هذه التكنولوجيا. ففي الماضي كان تصميم البرامج على أساس أن المستخدم يملك أقل المتطلبات لعملها ، مما نتج عنه برامج محدودة المستوى ، وقد أضطر المبرمجون لذلك حتى يتمكنوا من تشغيل برامجهم على أكثر عدد ممكن من أجهزة الكمبيوتر . أما الآن فقد انقلبت هذه القاعدة وأصبح المبرمجون يصممون برامجهم على أفضل المستويات لأن هذه التكنولوجيا تمكن أي مستخدم مهما كانت نوعية الجهاز لديه من تشغيل هذه البرامج.

: Retained Mode

تنقسم Direct3D إلى قسمين (أو كائنين) الأول يطلق عليه Retained-Mode أو RM ويسمح لنا بأن نستغل كل طاقة Direct3D في البرامج التي ننتجها بدون الحاجة إلى إنتاج مكتبة تهتم بتركيب الأجسام الثلاثية الأبعاد والتي سوف نستخدمها في برامجنا أو إلى المعلومات المتعلقة بتلك الأجسام أو إلى الحاجة لمعرفة الأنواع الجديدة من الأجسام الثلاثية المضافة ، فكل ذلك لا يهمننا في هذا القسم وبدل عنه نستطيع كمبرمجين أن نتحكم في كل ذلك. كأن نعبي جسم معين في الذاكرة مع الألوان المطلوبة لإعطائه المظهر المطلوب وذلك عن طريق أوامر بسيطة ومحدودة بالإضافة فإننا نستطيع أن نكبر ، نصغر ، ندير ، نضيء أو نغير أي جسم إلى جسم آخر في وقت حقيقي. كل ذلك بإمكاننا فعله بدون الحاجة إلى معرفة التفاصيل.

: Immediate Mode

والآخر Immediate-Mode أو IM وهو يستخدم من قبل المبرمجين الذين يريدون الحصول على تحكم أكبر في برمجة Direct3D والذين يريدون جلب مكتباتهم الجاهزة من نظام DOS إلى نظام Windows بأسهل طريقة ممكنة. هذا القسم يسمح لهم بأن يتدخلوا بشكل مباشر في عمل Direct3D وأجراء التغييرات التي يرونها مطلوبة لهم. ويختلف هذا النوع عن النوع الأول في أنه لا يوفر تعامل سهل مع الأجسام ثلاثية الأبعاد والتي هي أساس برمجة Direct3D. لذلك عند استخدام هذا النوع فإن Direct3D تتوقع من برنامجنا أن يتحمل عبئ التعامل مع هذه الأجسام. ومن حسنات هذا القسم أنه يوفر طريقة سهلة لتحويل البرامج التي تمت برمجتها على Dos إلى Windows ويسمح لنا بأن نتحكم في طريقة تعامل الكمبيوتر مع المشاهد في برامجنا مما يمكننا من استغلال كامل لكل ما هو جديد من مسرعات الأبعاد الثلاثية. سوف يكون تركيزنا في هذا الفصل على النوع الأول من Direct3D لأنه هو النوع المفضل لبرمجة هذه التكنولوجيا على الكمبيوتر ووجود النوع الثاني كان فقط في حالة استثنائية سبق وذكرناها. وقد كان RM مكتبة (المكتبة في هذه الحالة عبارة عن واجهات برمجية تعطينا تحكم كامل في البرنامج عن طريق بعض الأوامر البسيطة وتخفي عنا الكثير من الأجزاء البرمجية التي تمت برمجتها لنا ، وبذلك نستطيع استخدام هذه الأوامر مع التركيز على جودة البرنامج. كما كان الحال مع المكتبة Genesis 3D) بحد ذاتها تملك جميع الأوامر المطلوبة لإنشاء برامج ثلاثية الأبعاد ثم اشترتها شركة ميكروسوفت وسمتها Direct3DRM ، لذلك سوف نرى مدى قدرتها على إعطائنا التحكم الكامل على جهاز الكمبيوتر مع الإبقاء على سهولة الاستخدام إلى أقصى حد ممكن.

الواجهات البرمجية المستخدمة لـ Direct3D

من الناحية البرمجية Direct3DRM هي عبارة عن واجهات (أو كائنات) تمثل في مجموعها هذه التكنولوجيا ، والواجهة البرمجية هي الأوامر التي نكتبها في برنامجنا لإعطائنا النتيجة المرغوب فيها (بمعنى آخر هي أوامر الكائنات التي نستخدمها) و هنالك عدة واجهات برمجية توفر لنا تحكم كامل في البرنامج وهي كما يلي:

- الواجهة الرئيسية Direct3DRM
- واجهة الأداة DIRECT3DRMDEVICE
- الواجهة البديلة DIRECT3DWINDEVICE
- واجهة وجهة النظر DIRECT3DRMVIEWPORT
- واجهة الإطار DIRECT3DRMFRAME
- واجهة بنية الأجسام DIRECT3DRMMESHBUILDER
- واجهة الأجسام DIRECT3DRMMESH
- واجهة الأوجه المستخدمة DIRECT3DRMFACE
- واجهة خريطة الشكل الخارجي للأجسام DIRECT3DRMTEXTURES
- واجهة الغلاف الخارجي للأجسام DIRECT3DRMTEXTUREWRAP
- واجهة الإضاءة للشكل الخارجي DIRECT3DRMMATERIAL
- واجهة الإضاءة DIRECT3DRMLIGHT
- واجهة الظلال DIRECT3DRMSHADOW
- واجهة الحركة DIRECT3DRMANIMATION
- واجهة مجموعة الحركة DIRECT3DRMANIMATIONSET

الواجهة الرئيسية Direct3DRM

يتكون الكائن (object) Direct3DRM من Direct3D نفسها. ونقوم بتكوين الواجهة لهذا الكائن عن طريق الأمر Direct3DRMCreate() كما في المثال التالي :-

```
LPDIRECT3DRM d3drm;
Direct3dRMCreate(&d3drm);
```

في هذا المثال استخدمنا الأمر LPDIRECT3DRM ويقوم بإنشاء مؤشر (pointer) إلى واجهة الكائن Direct3DRM . ثم بعد ذلك يقوم الأمر الثاني في هذا المثال Direct3dRMCreate() بإنشاء الكائن المطلوب وكذلك ببداية عمل المؤشر. بعد أن تتم هذه العملية بنجاح يكون الكمبيوتر جاهزاً لاستخدام واجهة Direct3DRM. إنشاء واجهة الكائن Direct3DRM مهمة لأنها حلقة الوصل بين المبرمج و Direct3D فتعطينا القدرة على الاستغلال الكامل لها ونقوم بعد ذلك بإنشاء كائنات أو "واجهات" تمثل الواجهة الرئيسية وأغلبها يبدأ بكلمة "Create" أو أنشئ كما نراها هنا :-

- CreateAnimation() إنشاء الحركة
- CreateAnimationSet() إنشاء مجموعة الحركة
- CreateDeviceFromClipper() إنشاء كائن من ذاكرة النظام
- CreateDeviceFromD3D() إنشاء كائن من Direct3D
- CreateDeviceFromSurface() إنشاء كائن من ذاكرة الفيديو
- CreateFace() إنشاء وجه
- CreatFrame() إنشاء إطار

- **CreateLight()** إنشاء ضوء
- **CreateLightRGB()** إنشاء ضوء له لون معين يتكون من القيم (أحمر، أخضر، أزرق)
- **CreateMaterial()** إنشاء مادة معينة
- **CreateMesh()** إنشاء جسم ثلاثي الأبعاد
- **CreateMeshBuilder()** إنشاء باني الأجسام
- **CreateObject()** إنشاء كائن
- **CreateUserVisual()** إنشاء المستخدم المرئي
- **CreateShadow()** إنشاء ظلال
- **CreateTexture()** إنشاء خريطة الشكل الخارجي للأجسام
- **CreateTextureFromSurface()** إنشاء خريطة الشكل الخارجي للأجسام من ذاكرة الفيديو
- **LoadTexture()** تعبئة خريطة الشكل الخارجي للأجسام
- **CreateViewport()** إنشاء واجهة نظر
- **CreateWrap()** إنشاء الغلاف الخارجي

وليس من الضروري استخدام جميع هذه الكائنات أو الأوامر في برنامجنا ، وبعضها قريب الشبه من الآخر كما هو الحال مع الأمر `CreateLight()` و الأمر `CreateLightRGB()` فالفرق الوحيد بين الاثنين هو أن الأخير يعطينا القدرة على اختيار لون الضوء المستخدم. وفي لغة السي قد يستطيع كائن معين إنشاء كائن آخر ولذلك فإن في هذه الحالة الكائن `Direct3DRM` عادة يكون أول كائن يتم أنشأه في برنامجنا.

تعديل مسار البحث

أحد القواعد المهمة في واجهة Direct3DRM هي القدرة على تغيير أو تعديل مسار البحث وهذا المسار هو الذي تستخدمه Direct3D في التعرف على الملفات المطلوبة في البرنامج. ونستطيع أن نقوم بذلك التغيير أو الحصول على المسار عن طريق هذه الأوامر الثلاث :-

- **AddSearchPath()** أضف مسار البحث
- **GetSearchPath()** أحصل على مسار البحث
- **SetSearchPath()** أجعل مسار البحث

يكون مسار البحث في Direct3D عند بداية البرنامج `c:\dxsdk\sdk\media` ونستطيع تغيير هذا المسار عن طريق الأوامر السابقة. ولكن هذا التغيير يكون فعال فقط خلال عمل البرنامج. لذلك يجب الانتباه لأن المسار المطلوب قد لا يكون موجود على كمبيوتر المستخدم.

التحكم في سرعة البرنامج:

كيف نستطيع أن نتحكم في سرعة البرنامج ؟ أو بعبارة أخرى كيف نستطيع أن نتحكم في سرعة العرض ؟. Direct3DRM تحتوي على أمر `Tick()` وهو الأمر الذي يخول المبرمج على التحكم في سرعة العرض ، فكل مرة يُطلب فيها هذا الأمر تقوم Direct3D بتغيير الحركة على المشهد المعروض ، مثلاً في الأفلام السينمائية تقوم الكاميرا بعرض خمسة وعشرون صورة في الثانية وبذلك تظهر الحركة ، وهنا نستخدم

نفس الأسلوب فكلما نطلب الأمر Tick() يتم تجديد الشاشة بصورة جديدة فتظهر الحركة. ولذلك عندما يُطلب هذا الأمر بكثرة فإن الحركة تكون أسرع والعكس صحيح. وفي أغلب الأحيان فإن استخدام القيمة 1.0 هو الأفضل ، ولكن يمكن تغيير ذلك حسب حاجة البرنامج.

واجهته الأداة DIRECT3DRMDEVICE

واجهة الأداة لـ Direct3D هي عبارة عن مجموعة كائنات، وهذه الكائنات تعطينا النتيجة المرئية للبرنامج بحيث نرى المشاهد و الأجسام الثلاثية الأبعاد و التي يستخدمها البرنامج على الشاشة. و هناك أنواع مختلفة من الأدوات (devices) لذلك فإن Direct3D تستخدم عدد من تلك النوعيات. والبرامج تستطيع أن تختار نوع معين من الأدوات المتوفرة لها أو تستطيع أن تعتمد على Direct3D للحصول بشكل تلقائي عليها. وهناك نوعين رئيسيين من الأدوات هما النوعية البرمجية Software Devices التي تعتمد على البرنامج نفسه من خلال استغلال قوة المعالج المستخدم والنوعية الأخرى Hardware التي تستغل ما على الجهاز من مميزات مثل مسرع للأبعاد الثلاثية، وهذه النوعية لا تعمل إلا عند وجود هذه السرعات على الجهاز المستخدم.

ونوعية الأداة التي تمثل Direct3DRMDevice تعمل عن طريق أوامر الواجهة الرئيسية (Direct3DRM) وهناك ثلاث طرق لإنشاء الأداة :

1. الأولى هي عن طريق إنشاء كائن من نوع DirectDraw يسمى الكائن "clipper" أو الحاذف ومن ثم ننشئ الأداة عن طريق الأمر CreateDeviceFromClipper() الذي بدوره يقوم بإنشاء الواجهة من هذا الكائن الجديد، وهذه أسهل وآمن طريقة لإنشاء الأداة المطلوبة.

2. الثانية عن طريق إنشاء طبقة رئيسية لذاكرة الفيديو باستخدام DirectDraw ثم إنشاء طبقة أخرى ثانوية (back_buffer) حتى تسمح بالتغيير التصفيحي (page flipping) وهي عبارة عن طريقة خداع بصري لإتمام الحركة بشكل صحيح، ثم عن طريق الأمر CreateDeviceFromSurface() نقوم بإنشاء الأداة المطلوبة. وهذه

الطريقة المفضلة عندما نريد من برنامجنا العمل على الشاشة بكاملها (-full screen mode) وليس مجرد نافذة من نوافذ windows.

3. الثالثة هي استخدام القسم أو الكائن الأخر من Direct3D وهو Immediate Mode أو IM ثم بعد ذلك نقوم باستخدام الأمر (CreateDeviceFromD3D). عندما تتم إنشاء الأداة نستطيع أن نستخدمها في تحديد جودة شكل البرنامج. ويطلق على هذه العملية بـ Rendering وهي مشهورة في برامج إنتاج صور الأبعاد الثلاثية الثابتة والمتحركة ونبدأ في إظهار البرنامج على الشاشة.

بعد إنشاء الواجهة نستطيع استخدام الأوامر التي توفرها كما يلي :

- GetDirect3DDevice()
- GetHeight()
- GetTrianglesDrawn()
- GetViewports()
- GetWidth()
- GetWireframeOptions()
- Update()

الألوان:

في كلتا الحالتين سواء كان الأداة (device) برمجية أو تعتمد على السرعات الموجودة في الجهاز فإنها تتعامل مع الألوان بنفس الطريقة. وهناك نوعين من الإضاءة يمكن استخدامهما (RGB) ((أل R هو أول حرف من كلمة **RED** وتعني "أحمر" و أل G هو أول حرف من كلمة **GREEN** وتعني "أخضر" و أل B هو أول حرف من كلمة **BLUE** وتعني "أزرق" ، فعن طريق خلط هذه الألوان يحصل الكمبيوتر على أي لون مطلوب ولاحظ أن هناك ملايين الألوان في الطبيعة)) والأخر يطلق عليه Ramp . في النوع الأول (RGB) يستطيع إعطائنا أي ضوء ملون نرغب به أما (Ramp) ضوء غير ملون ، ولأنه لا يعطينا أي لون لذلك فهو أسرع من RGB ويمكننا كذلك تسميته "mono" أي

”أحادي” لأنه يتعامل مع الألوان كأبيض أو أسود ولاحظ أننا تحدثنا فقط عن الإضاءة فلو استخدمنا النوع Ramp فلا يعني بأننا لن نرى الألوان في برنامجنا ، بل يمكننا رؤية الألوان مادامت الأجسام المستخدمة تستخدم الألوان. ونستخدم الأمر `GetColorModel()` للحصول على النوع المستخدم.

أنواع أل Rendering

هناك عدة أوامر يجب أن نذكرها والتي نستطيع من خلالها التحكم في شكل النتيجة التي نراها على شاشة الكمبيوتر وهي كما يلي:

- `GetQuality()` أحصل على النوعية
- `SetQuality()` ضع النوعية
- `GetShades()` أحصل على قدر التظليل
- `SetShades()` ضع مقدار التظليل
- `GetDither()` أحصل على قدر الاهتياج
- `SetDither()` ضع مقدار الاهتياج
- `GetTextureQuality`
- `SetTextureQuality`

النوعية `Quality` :

عن طريق الأمرين `GetQuality()` و `SetQuality()` نستطيع التحكم في جودة النتيجة المرئية للبرنامج سواء كانت الأداة تستخدم نوع `flat` ، `Gourad` أو `Phong` ويمكن للأجسام أن تتحكم في جودها بشكل منفرد عن الأداة إذا كانت تلك الجودة أقل من النوعية التي نختارها للأداة (مثلا إذا أردنا أن نجعل برنامجنا يستخدم النوعية `flat` وهي ليست أفضل نوعية ولكنها سريعة وفي نفس الوقت نريد أن نظهر طائرة في مطار

مثلاً ولكن الطائرة والتي هي جسم ثلاثي الأبعاد تستخدم Phong فسوف نرى بأن كل شيء flat بما في ذلك الطائرة مع أنها تستخدم Phong وهو أفضل الأنواع، سوف نرى مثالا عمليا فيما بعد لتوضيح هذه العملية أكثر) وتكون جودة الأداة موضعه على flat إذا لم نغيرها.

التظليل Shades:

عن طريق الأمرين GetShades() و SetShades() نستطيع التحكم في عدد الظلال المستخدمة للون معين (مثلا اللون الأحمر قد يكون فاتح وقد يكون غامق وهناك درجات بين الفاتح والغامق وهذه الدرجات هي عدد الظلال المستخدمة للون) وطبعا عدد الظلال المستخدمة بشكل عام يعتمد على عدد الألوان التي يسمح بها البرنامج (قد تكون 8 bit (256 لون) وقد تكون 16 bit (65536 لون) وقد تكون 24 bit (16777216 لون) وقد تكون 32 bit (4294967296 لون). وتكون قدرة درجة التظليل هي 32 درجة ألا إذا غيرنا ذلك والتجربة هي أفضل طريقة لمعرفة أفضل درجة مناسبة لبرنامجك.

ملاحظة : درجة التظليل يجب أن تكون من مضاعفات العدد 2. مثل 4،6،8،10،.... الخ

الاهتياج Dither:

عن طريق الأمرين GetDither() و SetDither() نستطيع أن نشغل أو نطفئ هذه الميكانيكية. وهي طريقة نستطيع من خلالها أن نبرز عدد أكثر من الألوان عن ما هو متوفر فعلاً وهي مفيدة عندما نستخدم 8 bit ونريد أن نظهر أكثر من 256 لون وهذه الميكانيكية مفيدة حتى عندما نستخدم عدد كبير من الألوان مثل 16 bit وهي تكون مشغلة ألا إذا أطفأناها.

الواجهة البديلة DIRECT3DWINDEVICE

الكائن الذي يعمل كأداة لـ **Direct3D** هو عبارة عن كائن من نوع **COM** الذي باستطاعته استخدام أكثر من واجهة. وواجهة الأداة **Direct3DRMDevice** التي شرحناها هي عبارة عن نوعية واحدة من عدة أنواع من الأدوات التي نستطيع استخدامها. و **Direct3DWinDevice** هي عبارة عن أداة أخرى نستطيع استخدامها وهذه الأداة تابعة لنظام التشغيل ويندوز.

لأن الواجهة البديلة **Direct3DWinDevice** هي واجهة بديلة عن واجهة لأداة **Direct3D** لذلك فإن البرامج التي تريد استخدامها تقوم بذلك عن طريق استخدام الأداة المتوفرة على الجهاز في ذلك الحين عن طريق الأمر **QueryInterface()** أو كما يلي:

```
LPDIRECT3DRMWINDEVICE windev;
device->QueryInterface( IID_Idirect3DRMWinDevice, (void**)&windev
);
```

ونرى :

- الأمر **device** (أداة) يعني أن أننا سوف نستخدم مؤشر (pointer) إلى واجهة الأداة (**Direct3DRMDevice**).
- والأمر **QueryInterface()** أستخدم للحصول على مؤشر للواجهة البديلة **Direct3DRMWinDevice**
- أما الأمر **IID_Idirect3DRMWinDevice** فهو عبارة عن (**Unique GUID**) **Globally Identifier** للواجهة المطلوبة.

- أما الأمر الأخير WinDevice عبارة عن اختياريين الأول HandleActivate() و HandlePaint() وكلا الاثنان يرسل رسالة إلى Direct3D ليخبرها بأن رسالة من ويندوز وصلت. ثم أن البرنامج يقوم باستدعاء الأمر HandleActivate() كلما وصلت رسالة من الأمر WM_ACTIVATE أو استدعاء الأمر HandlePaint() كلما وصلت رسالة من الأمر WM_PAINT.

واجهة وجهة النظر DIRECT3DRMVIEWPORT

هذا هو الاسم الذي تستخدمه Direct3D للتعبير عن الكاميرا ولأننا نتعامل مع عالم ثلاثي الأبعاد لابد وأن تكون هناك كاميرا يستطيع بها المستخدم أن ينظر إلى ذلك العالم. واجهة النظر (viewport) هي التي تحدد الموقع الذي نستطيع من خلاله مشاهدة أي مشهد ونستطيع أن نحدد كذلك مقدار المساحة المرئية وبعدها المشاهد ونستطيع طبعاً أن نغير أو نحرك هذه الواجهة على حسب الحاجة. ونعبر عن واجهة النظر بالأمر التالي :

```
d3drm->CreateViewport( device, Camera, 0, 0, device->GetWidth(),
device->GetHeight(), &viewport );
```

الشرح:

كما نرى هنا المتغير d3drm هو عبارة عن مؤشر (pointer) إلى واجهة Direct3DRM والمتغير device هو عبارة عن مؤشر إلى واجهة Direct3DRMDevice أما المتغير Camera فهو عبارة عن frame الذي عن طريقه سوف نعرف موقع واجهة النظر. (ما هو frame؟؟)

ساحة الرؤية:

ساحة الرؤية (field-of-view (FOV)) يمكن التحكم بها عن طريق الأمر `SetField()` وتكون القيمة المبدئية لهذا المتغير 0.5 والقيم الصغرى تصغر زاوية ساحة الرؤية وتعطينا المؤثرات التصغيرية للعدسة والقيم الكبرى تكبر زاوية ساحة الرؤية وتعطينا المؤثرات التكبيرية للعدسة أما القيم السالبة فغير مسموح بها.

CLIPPING أو الحذف

واجهات النظر تتحكم في المقدمة الظاهرة للمشاهد والمؤخرة الغير ظاهرة للمشاهد عن طريق `3D clipping` ، بمعنى أننا نستطيع أن نستخدم الأمر `SetFront()` و الأمر `SetBack()` لكي نتحكم في تحديد المنطقة المرئية من الكاميرا بحيث أن جميع الأجسام الواقعة داخل هذه المنطقة تكون ظاهرة للمشاهد وجميع الأجسام والواقعة خارج هذه المنطقة لن تكون مرئية.

واجهة الإطار FRAME INTERFACE

تعتمد `Direct3D` كثيرا على هذه الواجهة لتتعرف على حركة الأجسام. جميع الكائنات في برامجنا مثل الكاميرات و الإضاءة و الأجسام ثلاثية الأبعاد لابد أن يكون لها موقع ، حركة واتجاه معين بحيث تتعرف عليها `Direct3D` وتتعامل معها على ذلك الأساس. ولكي نقوم بذلك فإن كل كائن (لاحظ أن كلمة كائن هنا تختلف عن كلمة كائن من الجهة البرمجية والتي نتحدث عنها كثيرا ، هنا تعني كل شئ يتحرك في محيط ثلاثي الأبعاد) يجب أن يكون له إطار بحيث يأخذ منه معلومات عن موقعة واتجاهه فإن تحرك الإطار تحرك الجسم معه.

تركيبية الإطار :

أن المشاهد في Direct3D هي عبارة عن إطارات متراكبة. وكل مشهد يحتوي على الإطار "الجزري" أو الإطار "الأم" وعلى عدد معين من الإطارات يطلق عليها "الأبناء" التابعة لذلك الإطار الجزري. وكل إطار "أبن" يستطيع أن يكون له إطارات "أبناء" خاصة به. وتعرف الإطار عن طريق الواجهة `Direct3DRMFrame` ونستطيع تكوينها عن طريق الأمر `CreateFrame()` كما نرى هنا:

```
LPDIRECT3DRMFRAME newframe;
d3drm->CreateFrame( parentframe, &newframe );
```

ونرى في هذا المثال مؤشر إلى إطار ، ثم إنشاء إطار عن طريق الأمر `CreatFrame()` ، والمتغير `parentframe` ويعني الإطار الجزري أو الإطار الأم هو عبارة عن مؤشر إلى الإطار الذي سوف يكون الإطار الأم للإطار الجديد ولو استخدمنا الأمر `NULL` بدلا عنه فسوف ينشئ إطار جزري للمشهد ككل. ويجب علينا معرفة أن الإطارات الأبناء تأخذ جميع صفات الإطارات الأم، فمثلا عندما ننشئ إطار أبن من إطار أم فإن هذا الابن يكون تابعا لها فلو تحركت الأم فإن الابن سوف يتحرك معها وهذا يعطينا إمكانية هائلة في إنشاء الحركة وتوابعها. لنأخذ مثال على ذلك لو أردنا إنشاء طائرة مروحية تطير من النقطة ألف إلى النقطة باء بحيث يكون لها مروحة علوية متحركة ومروحة خلفية متحركة أيضاً فإننا سوف ننشئ إطار أم لجسم الطائرة ثم إطار أبن لكل مروحة بحيث نحرك المروحة عن طريق تحريك إطارها بشكل دائري ولأن إطارات المراوح أبناء لإطار جسم الطائرة فإنهم سوف يتبعون إطار جسم الطائرة إذا تحرك أو بعبارة أخرى سوف يقلدون حركة الأم. وحركة الإطارات واتجاهاتها ليست محدودة بحركة الأم ، فبالإمكان استخدام أي إطار في المشهد ليكون الأم لأي إطار

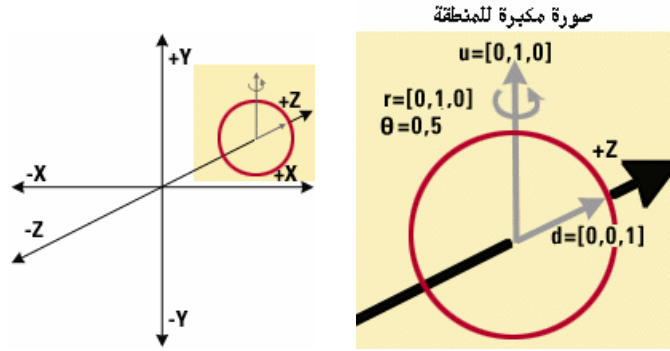
آخر وبالتالي نستطيع أن نتحكم بحرية في حركة الإطارات وبالتالي نتحكم في حركة الكائنات في المشهد.

موقع الإطارات

أن واجهة الإطارات (**Direct3DRMFrame**) توفر لنا الكثير من الأوامر التي عن طريقها نستطيع أن نتحكم في مواقع الإطارات كما يلي:

- **AddRotation()** أضف درجة الدوران
- **AddScale()** أضف درجة الحجم
- **AddTranslation()** أضف الموقع
- **GetOrientation()** أحصل على الاتجاه
- **GetPosition()** أحصل على الموقع
- **LookAt()** أنظر إلى
- **SetOrientation()** ضع درجة الاتجاه
- **SetPosition()** ضع الموقع

توفر لنا واجهة الإطار الأوامر المطلوبة لتحريك الإطارات وتعديل وجهتها وموقعها طوال مدة عمل البرنامج ، بحيث تقوم هذه الأوامر بفحص حالة الإطارات مع كل تجديد للمشهد.



وكما نلاحظ أن هذه الأوامر تعطينا تحكم كامل في الإطارات ،الأمران `GetPosition()` و `SetPosition()` يعطيانا القدرة على تحديد موقع الإطار. والأمران `GetOrientation()` و `SetOrientation()` يعطيانا القدرة على تحديد جهة الإطار (لاحظ المتجهين d و u في الشكل). الأمران `GetRotation()` و `SetRotation()` يستعملان لتغيير درجة دوران الإطار. أما الأمر `LookAt()` فإنه يقوم بتوجيه الإطار إلي وجه (كل جسم ثلاثي الأبعاد يتكون من أوجه متشابهة تعطي في مجموعها شكل الجسم ككل) معين أو إلى إطار آخر. ومن أفضل الاستخدامات لهذا الأمر هو توجيه الكاميرا أو الإضاءة إلى جسم متحرك. وكل من الأوامر `SetPosition()` و `SetOrientation()` و `LookAt()` تقوم بعملها بدون أن تكثرث لحالة الإطار السابقة أما الأوامر `AddTranslation()` و `AddRotation()` و `AddScale()` تقوم بتعديل حالة الإطار مع الأخذ في عين الاعتبار الحالة السابقة للإطار، مثلاً `AddTranslation()` تقوم بتحريك الإطار من الموقع السابق إلى الموقع الجديد و `AddScale()` تقوم بتغيير حجم الكائن ابتداءً بحجمه الحالي إلى أن يصل للحجم المطلوب.

واجهة بنية الأجسام DIRECT3DRMMESHBUILDER

الواجهة Direct3DRMMeshBuilder تعطينا الأدوات الأزممة للتعامل مع الأجسام ثلاثية الأبعاد في برنامجنا وهناك ما يقارب 38 أمر لإنشاء و تعديل هذه الأجسام. بنية الأجسام تستخدم لبناء الأجسام وليست هي جسم بحد ذاتها ولكن يمكن استخدامها لتكون بديل لتلك الأجسام، فباستطاعتنا إضافة بنية الأجسام ككائن مرئي ولكن في تلك الحالة فإن هذه البنية تقوم باستخدام جسم ثلاثي الأبعاد بشكل داخلي (نعم ، تحتاج أن تأخذ نفس عميق 😊) وننشئ بنية الأجسام عن طريق الأمر CreatMeshBuilder() كما يلي:

```
LPDIRECT3DRMMESHBUILDER meshbuilder;  
d3drm->CreateMeshBuilder( &meshbuilder );
```

التعبئة (Loading) و الحفظ (Saving) :

الأمر Load() يعطي القدرة لبنية الأجسام بتعبئة الأجسام المطلوب استخدامها من القرص (سواء كان القرص المدمج أو القرص الصلب) أو من موارد البرنامج أو من الذاكرة. وكذلك يمكننا استخدام هذا الأمر لتعبئة الصور "Texture" الخارجية للأجسام (أي خريطة الشكل الخارجي للأجسام) إذا كان مسار البحث قد تم تعريفه لـ Direct3D. ويسمح هذا الأمر باستخدام خاصية Callback كل ما أردنا تعبئة خريطة الشكل الخارجي للأجسام فنحصل على القدرة بأن نتحكم في تغيير موقع التعبئة عن ما هو عليه في مسار البحث سواء كان هذا الموقع على قرص أو في موارد البرنامج.

أما الأمر Save() فإنه يسمح لنا بحفظ الأجسام كملف مكتوب "text" أو كملف "binary" أو كملف مضغوط.

أنواع أل Rendering

توفر لنا هذه الواجهة عدد من الأوامر التي نستطيع من خلالها التحكم في النتيجة النهائية لشكل الأجسام ، ويجب ملاحظة أن هذه الأوامر تختلف عن الأوامر التي سبق ذكرها (راجع أوامر واجهة الأداة) للبرنامج ككل ففي هذه الحالة فإننا نهتم فقط بالأجسام التي نتعامل معها بشكل منفرد و هناك عدة أوامر نذكرها كما يلي:

- GetQuality() أحصل على النوعية
- SetQuality() ضع النوعية
- SetColor() ضع اللون
- SetColorRGB() ضع لون معين يتكون من القيم (أحمر، أخضر، أزرق)
- SetTexture() ضع خريطة الشكل الخارجي
- GetTextureCoordinates() أحصل على إحداثيات خريطة الشكل الخارجي
- SetTextureCoordinates() ضع إحداثيات خريطة الشكل الخارجي
- SetTextureTopology()
- SetPerspective() ضع وجهة النظر
- GetPerspective() أحصل على وجهة النظر

عن طريق الأمرين GetQuality() و SetQuality() نستطيع التحكم في جودة الأجسام المرئية سواء كانت من نوع Gourad، flat، wireframe أو Phong ويمكن للأجسام أن تتحكم في جودة أشكالها إذا كانت تلك الجودة مثل أو أقل من النوعية التي نختارها للبرنامج (مثلاً إذا كان برنامجنا يستخدم النوعية الإطار السلكي (wireframe) وهي ليست أفضل نوعية ولكنها سريعة وفي نفس الوقت نريد أن نظهر طائرة في مطار مثلاً ولكن الطائرة والتي هي جسم ثلاثي الأبعاد تستخدم Phong فسوف نرى بأن كل شيء يظهر بجودة wireframe بما في ذلك الطائرة حتى لو أنها تستخدم Phong وهو أفضل الأنواع، سوف نرى فيما بعد مثلاً عملياً لتوضيح هذه العملية أكثر) وتكون الجودة موضعه على flat إذا لم نغيرها.

الأمرين SetColor() و SetColorRGB() يمكن استعمالهما لإعطاء ألوان معينة للأوجه في الأجسام ولاحظ عدم وجود الأمر GetColor() وذلك لأنه لا يوجد ضمان بأن جميع الأوجه على أي جسم سوف تكون بنفس اللون.

أما الأمر SetTexture() فنستطيع من خلاله أن نضع خريطة لصورة معينة بحيث تعطي شكل معين لباني الأجسام والأوامر

- GetTextureCoordinates()
- SetTextureCoordinates()
- SetTextureTopology()

نستطيع من خلالها التحكم في تلك الصورة بحيث نضعها على الشكل الصحيح وفي الحقيقة بأن هذه الأوامر وضعت حتى تعطينا تحكم كامل على شكل الأجسام والطريقة المفضلة هي استخدام طريقة الصور الملفوفة بدلاً عن هذه الأوامر وسوف نتكلم عن هذه الطريقة فيما بعد.

وهناك ميكانيكية في هذه الواجهة تقوم بتصحيح وضع الصور على الأجسام بحيث نتأكد بأن كل شيء يظهر كما هو مطلوب فمن خلال الأمرين GetPerspective() و SetPerspective() نستطيع تشغيل أو توقيف عمل هذه الميكانيكية، ولكنها أيضاً

تزيد من العمليات الحسابية للكمبيوتر وبذلك تقلل من السرعة النهائية للبرنامج، وهذه الميكانيكية مفضلة عندما يكون المشاهد قريب من جسم معين ويمكن توقيفها عندما تكون الأجسام بعيدة عن المشاهد. وكمبرمجين سوف نلاحظ ذلك عندما نقوم بتجربة البرنامج فإننا سوف نرى أن الصورة الخارجية ليست ثابتة على الجسم ولذلك نقوم بتشغيل هذه الميكانيكية أو توقيفها.

التحكم في الأوجه

أن واجهة بنية الأجسام (Meshbuilder) تسمح لنا أيضا بأن نتحكم في الأوجه للأجسام فمثلا مجموعة الأوجه التي تمثل جسم معين نستطيع أن نحصل على قائمة بها عن طريق الأمر GetFaces() أو عددها عن طريق الأمر GetFaceCount() ويمكننا بناء (عن طريق الأمر CreateFace()) و إضافة (عن طريق الأمر AddFace() أو AddFaces()) أي وجه أو أوجه إلى بنية الأجسام.

التحكم بالنقاط (VERTEX)

وتعطينا هذه الواجهة كذلك القدرة على التحكم بنقاط أي جسم (كل جسم يتكون من عدد من الأوجه وكل وجه يتكون من عدد من النقاط المتصلة) فيمكننا الحصول على عدد النقاط لأي جسم عن طريق الأمر GetVertexCount() و كل نقاط الجسم يمكننا الحصول عليها وتغييرها أو إنسابها عن طريق الأمرين GetVertices() و SetVertices() ويجب ملاحظة أهمية هذا التحكم لأننا نستطيع تغيير شكل أي جسم عن طريق التحكم بنقاطه.

الحركة والحجم

الكائنات تستخدم الإطارات في حركتها وموقعها ولكنها أيضاً تستطيع أن تتحكم في طريقة تعاملها مع الإطارات الملحقه بها فلأمر Translate() يعطينا القدرة على موازنتها (تحريكها) مما يعطينا القدرة أيضاً على ربط أكثر من جسم بإطار واحد فقط من دون أن يحتل أي جسم نفس المكان لجسم آخر. والأمر Scale() يعطينا القدرة على التحكم بحجم أي جسم بحيث يكون تغير الحجم على أحد الإحداثيات التابعة لذلك الجسم (يعنى ذلك بأننا نستطيع أن نغير حجم الجسم على أي إحداثي كالإحداثي السيني مثلاً).

سرعة الأداء

بسبب قوة وسهولة استخدام هذه الواجهة (Direct3DRMMeshBuilder) فإننا نستطيع استخدامها في مهام عديدة ولكن عند مقارنتها بسرعة الأداء فإن هذه الواجهة تفتقد لبعض السرعة في بعض الأحوال فعندما نستخدم هذه الواجهة في تعبئة وتعديل جسم معين ثم نستخدم النتيجة (أي الجسم الناتج بعد التعديل) في برنامجنا بدون تعديلات إضافية متكررة فإن هذه الواجهة تعمل بشكل جيد ولكن عندما نريد تغير مستمر في جسم معين (كأن نغير لونه و شكله و نقاطه... الخ) خلال عمل البرنامج فإن هذه الواجهة تعمل بشكل غير جيد ولكن يمكننا استعمال بدائل عن ذلك سوف نتحدث عنها في الواجهة القادمة (واجهة الأجسام).

واجهة الأجسام DIRECT3DRMMESH

تكمّن قوه هذه الواجهة (Direct3DRMMesh) في سرعتها ولكننا تفتقد إلى السهولة التي رأيناها في واجهة بنية الأجسام السابقة. ونستطيع أن ننشئ الأجسام عن طريق الأمر CreateMesh() في الواجهة الرئيسية (Direct3DRM) أو عن طريق الأمر CreateMesh() في واجهة بيئة الأجسام (Direct3dRMMeshBuilder) .

مجموعات الكائنات (أو مجموعات الأجسام ثلاثية الأبعاد) :

- أضف المجموعة AddGroup()
- أحصل على المجموعة GetGroup()
- أحصل على لون المجموعة GetGroupColor()
- أحصل على عدد المجموعة GetGroupCount()
- أحصل على خريطة المجموعة GetGroupMapping()
- أحصل على إضاءة الشكل الخارجي للمجموعة GetGroupMaterial()
- أحصل على جودة المجموعة GetGroupQuality()
- أحصل على خريطة الشكل الخارجي للمجموعة GetGroupTexture()
- ضع لون المجموعة SetGroupColor()
- ضع لون معين للمجموعة يتكون من القيم (أحمر، أخضر، أزرق) SetGroupColorRGB()
- ضع خريطة للمجموعة SetGroupMapping()
- ضع إضاءة الشكل الخارجي للمجموعة SetGroupMaterial()
- ضع جودة المجموعة SetGroupQuality()
- ضع خريطة الشكل الخارجي للمجموعة SetGroupTexture()

ونستخدم هذه الأوامر للتحكم بمجموعات الأجسام ويجب أولاً أن ننشئ المجموعة باستخدام الأمر AddGroup() بحيث نستخدم النقاط و الأعمدة و إحداثيات خريطة الشكل الخارجي للأجسام مع هذا الأمر لإنشاء المجموعة وبعد إنشاء أي مجموعة نستخدم هذه الأوامر للتحكم بها.

التحكم بالنقاط (Vertex)

أن واجهة الأجسام تعطينا القدرة في التحكم بنقاط أي جسم ثلاثي الأبعاد عن طريق الأمرين GetVertices() و SetVertices() بحيث نستطيع تغيير موضعها فلأمر الأول يعطينا المعلومات المطلوبة عن النقطة وعندما نحصل على تلك المعلومات (مثل الموقع و الإحداثيات ... الخ) نستطيع تغييرها باستخدام الأمر الثاني.

إنشاء جسم ثلاثي الأبعاد عن طريق استخدام واجهة بنية الأجسام

نستطيع استخدام الواجهة السابقة (واجهة بنية الأجسام Meshbuilders) في إنشاء كائن ثلاثي الأبعاد في هذه الواجهة (واجهة الأجسام) وكما عرفنا سابقاً أن واجهة بنية الأجسام تستخدم في تعبئة الأجسام من القرص إلى الذاكرة (حتى نستطيع تعديل الألوان والشكل الخارجي والأوجه والنقاط والأعمدة لتلك الأجسام) وكذلك نستطيع أن نستخدمها في إنشاء الأجسام عن طريق الأمر CreateMesh() بحيث نقوم عن طريقة بإنشاء كائن من نوع Direct3DRMMesh وهذه طريقة مريحة فعلاً ولكنها تعني بأن جميع الأوجه في ذلك الجسم الناشئ سوف تكون في مجموعة واحدة فقط

ولو أردنا تقسيمها إلى عدة مجموعات فإن ذلك يجب أن يتم عن طريق استخدام الأمر
 . AddGroup()

واجهة الأوجه المستخدمة DIRECT3DRMFACE

جميع الأجسام التي نتعامل معها على الكمبيوتر هي عبارة عن مجموعة أوجه تشكل في مجموعها الجسم الثلاثي الأبعاد ، وفي أغلب الأحيان سوف نقوم بتعديل تلك الأوجه بدلا عن إنشائها. وفي واجهة بنية الأجسام (MeshBuilder) تعرفنا على بعض الأوامر التي تؤهلنا للتحكم بالأوجه مثل الأمر GetFaces() ومع هذا فإن باستطاعتنا إنشاء الأوجه وأضافتها إلى واجهة بنية الأجسام وعلى أي حال فإن بإمكاننا أن نتحكم في أي وجه عن طريق التحكم في لونه بالأمرين GetColor() و SetColor() أو شكله الخارجي أو نقاطه وجميع الأوجه تكون ممثلة عن طريق هذه الواجهة (واجهة الأوجه المستخدمة أو Direct3DRMFace).

أوامر خريطة الشكل الخارجي للوجه:

- GetTexture()
- SetTexture()
- GetTextureCoordinates()
- SetTextureCoordinates()
- GetTextureTopology()
- SetTextureTopology()
- GetTextureCoordinateIndex()
- SetTextureCoordinates()

تعطينا هذه الأوامر تحكم كامل في الشكل الخارجي لأي وجه بحيث نستطيع عن طريقها أن نضع خريطة لشكل معين على أي جسم. و سوف نتكلم عن هذه الأوامر بشكل أكثر دقة في واجهة خريطة الشكل الخارجي.

أوامر إضاءة الشكل الخارجي للوجه:

- احصل على إضاءة الشكل الخارجي للوجه `GetMaterial()`
- ضع إضاءة الشكل الخارجي للوجه `SetMaterial()`

نستطيع من خلال هذه الأوامر وبالتعاون مع أوامر خريطة الشكل الخارجي أن نضع إضاءة الشكل الخارجي للأوجه. و كذلك سوف نتكلم عن هذه الأوامر بشكل أكثر دقة في واجهة إضاءة الشكل الخارجي.

أوامر نقاط الأوجه

- أضف النقطة `AddVertex()`
- احصل على النقطة `GetVertex()`
- احصل على رقم النقطة `GetVertexCount()`
- احصل على فهرس النقطة `GetVertexIndex()`
- `GetVertices()`

كما عرفنا سابقا بأن كل وجه يتكون من عدة نقاط متصلة ببعضها البعض بحيث تعطي في مجموعها شكل الوجه ولذلك فإن تغيير أي نقطة في أي وجه سوف يغير ذلك الوجه وعندما تزيد عدد النقاط في الوجه فإنه يأخذ وقت أكثر من الكمبيوتر

لرسمة ولذلك فإن Direct3DRM تقوم بتعامل مع ثلاث نقاط فقط لكل وجه وهذه من مميزات هذا النوع من Direct3D.

واجهة خريطة الشكل الخارجي DIRECT3DRMTEXTURES

نستخدم هذه الواجهة لتحديد الأشكال الخارجية للأجسام مثلا مستطيل عليّة شكل الحجارة يظهر وكأنه جدار من حجر ويمكننا كذلك إضافة أشكال معينة للأوجه كما سبق وشرحنا. ومن خلال هذه الواجهة نستطيع أن نضع خلفية معينة لأي مشهد في برامجنا الثلاثية الأبعاد. وهناك ثلاث طرق يمكننا بها استخدام الشكل المطلوب. الأولى عن طريق وضع الشكل المطلوب على ملف من نوع BMP وهو النوع المعروف لويندوز وهو موجود في كل برامج ويندوز المختصة في الرسم بما في ذلك برنامج paint. أو استخدام الملفات من نوع PPM. أو أخيرا عن طريق موارد البرنامج (Program's resources).

صنع الشكل الخارجي

أسهل طريقة لصنع الشكل الخارجي في البرنامج هو عن طريق الأمر
Direct3DRMLoadTexture(). ونستخدمه كما يلي:

```
LPDIRECT3D texture;  
d3drm->LoadTexture("texture.bmp", &texture );
```

فمن خلال هذا الأمر نخبر الكمبيوتر بوجود ملف من نوع Bmp (أيضا بالإمكان استخدام النوع PPM) أسمه texture.bmp فتقوم Direct3D عن طريق الأمر Direct3DRM LoadTexture() بتعبئة الملف في الذاكرة استعدادا لاستخدامه. وكما ذكرنا أننا أيضا

نستطيع استخدام موارد البرنامج (Program's resources) في وضع الشكل المطلوب عن طريق الأمر LoadTextureFromResource() كما يلي :

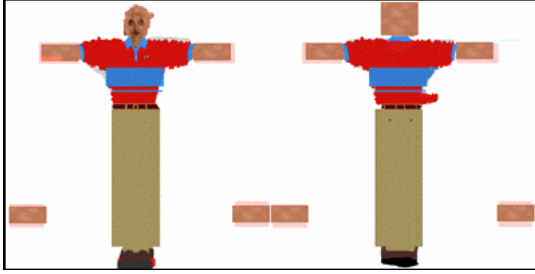
```
LPDIRECT3DRMTEXTURE texture;
HRSRC id = FindResource(NULL,
MAKEINTRESOURCE(IDR_SAMPLETEXTURE), "TEXTURE");
d3drm->LoadTextureFromResource( id, texture );
```

فكما نرى هنا أن الأمر Direct3DRM LoadTextureFromResource() يأخذ رقم معرف (كل شيء في موارد البرامج له رقم خاص به يسمى id) اختصار كلمة identification وتعني تحقيق الهوية) ويتعرف عليه البرنامج بذلك الرقم وفي هذه الحالة فإنها صورة لشكل معين) و يضعه في الذاكرة لاستخدامه عن طريق البرنامج. وتتخذ Direct3D من الطبقة التي تكونها DirectDraw في الذاكرة كصورة لها وتعرف هذه الطبقة بـ Surface ونستطيع نحن كمبرمجين أن نستخدم هذه الطبقة كذلك عن طريق الأمر LoadTextureFromSurface().

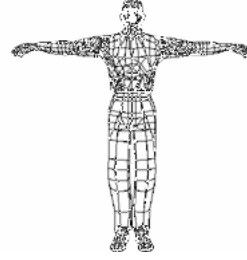
واجهته الغلاف الخارجي للأجسام DIRECT3DRMTEXTUREWRAP

هذا الواجهة عبارة عن كائن يقوم بوضع خريطة الشكل الخارجي للأجسام على الأجسام ثلاثية الأبعاد وبذلك يملك أوامر يتحكم باستخدامها في طريقة الوضع -سواء كانت سطحية (flat) أم أسطوانية (cylindrical) أم كروية (spherical)- و اتجاه الخريطة (الخريطة هي عبارة عن صورة للشكل الخارجي) على الجسم و حجمها ومركزها. والغلاف الخارجي للأجسام ينشأ باستخدام هذه الأوامر سواء لباني الأجسام أو لجسم ثلاثي الأبعاد. ونستطيع تحديد كيفية وضع خريطة الشكل الخارجي لكل وجه على ذلك الجسم ، وبذلك يوفر علينا الكثير من الوقت في وضع تلك الخريطة لأن بعض الأجسام قد يملك آلاف الأوجه.

وتتمثل هذه الواجهة `Direct3DRMTextureWarp` الغلاف الخارجي للأجسام وننشئها عن طريق الأمر `CreatWarp()` وهو أحد أوامر `Direct3DRM`.



خريطة الشكل الخارجي للجسم



الإطار السلكي لجسم ثلاثي الأبعاد



النتيجة النهائية بعد إضافة خريطة الشكل الخارجي

واجهة الإضاءة للشكل الخارجي `DIRECT3DRMMATERIAL`

تختلف هذه الواجهة عن واجه الإضاءة والتي سنتعرف عليها بعد قليل ، حيث أن هذه الواجهة تهتم بالشكل الخارجي للأجسام وليس بالإضاءة بشكل عام للمشاهد. وتحدد هذه الواجهة تأثير الإضاءة على الأوجه. ويمكننا إنشاء هذه الواجهة (`Direct3DRMMaterial`) عن طريق استخدام أحد أوامر `Direct3DRM` وهو `CreateMaterial()` وتعطينا هذه الواجهة القدرة على اختيار :

- قوة لمعان الإضاءة
- و لون لمعان الإضاءة
- و لون الإضاءة المنبعثة (emissive light color).

قوة لمعان الإضاءة

لمعان الإضاءة هي الإضاءة المنعكسة من الوجه ، والتي تحدد مدى تأثر ذلك الوجه بالإضاءة و قرية منها. ونحصل عادة على انطباع معين عن جسم ما من خلال مقدار تأثره بالضوء ، مثلا الأجسام اللامعة تعكس الضوء أكثر من الأجسام الداكنة. وقوة لمعان الضوء تكون أكبر في الأجسام اللامعة عنه في الأجسام المطاطية مثلاً. ويمكننا التحكم في تلك القيم من خلال الأمرين :

- GetPower()
- SetPower()

بإعطائها قيم معينة فالقيم الكبيرة تعطي قوة لمعان أكبر (لاحظ أن أكبر بمعنى أكبر في المساحة وليس في الشدة) والعكس صحيح.

لون لمعان الإضاءة

ويمكننا كذلك التحكم في لون اللمعان عن طريق الأمرين :

- GetSpecular()
- SetSpecular()

ويكون لون الإضاءة أبيض ما لم نغيره عن طريق هذين الأمرين.

لون الإضاءة المنبعثة

الإضاءة المنبعثة هي الإضاءة الناتجة من الجسم نفسه وليست منعكسة من ضوء آخر ، وعادة لا تكون الأجسام لها إضاءة منبعثة ولكن أحيانا من المفيد استخدام هذا النوع من الإضاءة عندما نريد أن نجعل جسم معين يصدر ضوء ، كالمصباح مثلا. وعندما نستخدم النوع Ramp من أنواع الإضاءة فإننا لا نستطيع أن نستخدم الضوء الملون ولكن الضوء المنبعث ليس له علاقة بواجهة الإضاءة لذلك يمكن استخدامه بأي لون في كلا النوعين Ramp و RGB.

واجهة الإضاءة DIRECT3DRMLIGHT

تختلف هذه الواجهة عن الواجهة السابقة (واجهة الإضاءة للشكل الخارجي) في أنها المسئولة عن الإضاءة في المشهد ككل. وتستطيع Direct3D عن طريق هذه الواجهة أن تستخدم خمسة أنواع من الإضاءة كما يلي :

1. الإضاءة الطيفية Ambient Light

2. الإضاءة النقطية Point Light

3. الإضاءة المتجهة Directional Light

4. الإضاءة المتوازية Parallel Light

5. الإضاءة المسلطة Spot Light

وسوف نشرح بعد قليل كل نوع بالتفصيل ، وكل نوع من هذه الأنواع يمكن إعطائه موقع واتجاه معين في المشهد ولكن بعضها لا يهتم بالاتجاه ويمكن كذلك إعطائها لون

معين بحيث تضيء بذلك اللون. ولأن كل نوع يختلف عن الآخر فكل نوع له طريقة مختلفة لاستخدامه في المشهد.

تتمثل الإضاءة عن طريق واجهة Direct3DRMLight (بمعنى واجهة الإضاءة) و ننشأها عن طريق الأمرين Direct3DRM CreatLight() و CreateLightRGB() وكلا الأمرين يطلب منا أن نذكر اللون المطلوب للإضاءة وكذلك نوع الإضاءة ويمكننا تغيير ذلك في أي وقت (بمعنى أننا نستطيع أن نغير لون الإضاءة أو نوعها في أي وقت نشاء) ومصادر الإضاءة تأخذ مكانها واتجاهها من الإطارات لذلك يجب ربط الإضاءة بإطار معين قبل أن يتسنى لنا استخدامه. وتوفر لنا واجهة الإطارات الأمر AddLight() لكي نقوم بذلك.

الإضاءة الطيفية

هذه هي اسهل الأنواع وأكثرها استخداما لأنها تأخذ فقط متغير واحد وهو متغير اللون (عن طريق هذا المتغير نستطيع أن نحدد اللون للإضاءة). ومع أننا يجب أن نربط هذه الإضاءة مع إطار (راجع واجهة الإطارات لمعلومات أكثر عنها) معين ألا إنها لا تهتم بموقع أو اتجاه ذلك الإطار. ونستطيع أن نستخدم أي عدد من مصادر الإضاءة في نفس المشهد وتكون لون وقوة الإضاءة الطيفية في مشهد واحد مجموع شدة وألوان الإضاءات في ذلك المشهد فعندما يكون لدينا مثلا ثلاث أضواء طيفية: أحمر و أخضر و أزرق فإن مجموع تلك الأضواء تعادل إضاءة طيفية واحدة ذات لون ابيض.

الإضاءة النقطية

كما هو الاسم فإن هذه الإضاءة عبارة عن نقطة تصدر ضوء في جميع الاتجاهات - تشبه الإضاءة الشمسية في برامج الأبعاد الثلاثية - ونستطيع أن نمثلها بالشمس حيث أن الضوء يكون متجها في كل الزوايا. لذلك فاتجاه هذه الإضاءة في المشهد غير مفيد و لكن موقعها يحدد مصدر أضائها.

الإضاءة المتجهة

الإضاءة المتجهة هي الإضاءة ذات اتجاه معين وهذا النوع من الإضاءة لها وجهه ولكن ليس لها مكان محدد لأنها تصدر أشعة ضوئية في خط مستقيم فلا نستطيع تحديد بداية تلك الأشعة ولكن لها اتجاه معين يكون قد حدد سلفا من خلال الإطار المربوط بها. وكذلك فإن هذه الإضاءة تعتبر أفضل من الإضاءة النقطية من حيث سرعة الأداء (في كل الأحوال يجب أن نبقى هذه القاعدة في أذهاننا بأن كلما زاد تعقيد المشهد من خلال زيادة الأجسام ثلاثية الأبعاد المستخدمة و عدد الإضاءات فيه فإن سرعته سوف تقل وذلك لأن الكمبيوتر عليه أن يقوم بعمليات حسابية كثيرة و التجربة هي الطريقة الوحيدة للحكم على مدى تعقيد أي مشهد).

الإضاءة المتوازية

تشبه هذه الإضاءة كثيرا الإضاءة النقطية مع اختلاف واحد ، هو أن هذا النوع من الإضاءة له اتجاهين منعكسين (بمعنى أنه يطلق شعاعين للضوء يكونا عكس بعضهما في الاتجاه) بدل اتجاه واحد للأشعة كما في الإضاءة النقطية. ويجب ملاحظة أن هذا النوع من الإضاءة يتأثر بالموقع والاتجاه فالموقع يحدد النقطة التي تفصل كل من الشعاعين والاتجاه يحدد وجهة الأشعة.

الإضاءة المسلطة

هذا النوع من الإضاءة يشبه الكشاف بشكلها المخروطي بحيث تبدأ كنقطة صغيرة في مصدر الضوء وتكبر بشكل دائري كلما ابتعد عن المصدر مما يعطي الشكل المخروطي ، وأقرب مثال حي على ذلك هو كما ذكرنا كشاف ضوء. وتستخدم هذه الإضاءة (كما في الأنواع الأخرى) الإطار لكي تحدد موقعها أو مصدرها وكذلك اتجاهها.

و الأشعة الصادرة من هذه الإضاءة تتكون من مخروطين إحداهما في الآخر ونحدهما من خلال زاويتين ، الزاوية المركزية Umbra وهي الزاوية التي تحدد المخروط الداخلي ويكون مركز الإضاءة حيث أن الضوء في أشد درجاته والزاوية المحيطة Penumbra وهي الزاوية التي تحدد المخروط المحيط بالمخروط الداخلي و يحدد هذا المخروط نهاية الضوء. والإضاءة في المنطقة بين المخروطين تختلف في درجاتها حيث يكون الضوء شديد (يتكون من اللون المستخدم) بقرب المخروط الداخلي ويبدأ في

الاضمحلال كلما اقتربنا من المخروط الخارجي إلى أن يصل إلى الانعدام (أو اللون الأسود) عند حدود المخروط الخارجي.
ونتحكم في الزاويتين المركزية و المحيطية عن طريق الأمرين SetUmbra() و SetPenumbra() وهما طبعا جزء من واجهة الإضاءة (Direct3DRMLight).

واجهة الظلال DIRECT3DRMSHADOW

هذه هي الواجهة التي تتعامل مع ظلال الأجسام ، من التجربة نعرف أن الظلال تعطي المشهد -أي مشهد كان- شيء من الواقعية . ولكن Direct3D صممت على أساس السرعة وكان ذلك طبعا على حساب أشياء أخرى مثل الظلال. مع الأسف Direct3D لا تعطينا إمكانية الحصول على ظل جسم معين في المشهد ، لذلك كان علينا إيجاد حل بديل لهذه المشكلة وكان الحل هو إضافة أجسام أخرى تحمل شكل الظل المطلوب لكي تظهر وكأنها ضلال. ولكن هذه الظلال لها مواقع تقريبية و كذلك لها بعض الإمكانيات المحدودة. وعندما نريد أن نستخدم هذه الظلال فإننا نحتاج إلى ما يلي :

- الجسم المنتج للظل
- الإضاءة المؤثرة في الجسم لإعطاء الظل
- السطح العاكس للظل

ونرى الإمكانيات المحدودة لهذه الطريقة من خلال حاجتها إلى معرفة مصدر الضوء ، فكل ضوء يفترض أن يعكس الضلال عليه أن يكون له ظل خاص به ، وذلك من خلال جسم ثلاثي ملحق به. وكذلك حاجتنا إلى معرفة موقع الظل على السطح العاكس له يعني عدم قدرتنا على عكس ظل لجسم مركب (أي جسم يحتوي على أكثر من عضو

بحيث أن كل عضو يعتبر جسم ثلاثي الأبعاد بحد ذاته) وكذلك فالسطح ذاته يجب أن يكون مستوي حتى يتسن لنا عكس الظل عليه بشكل جيد.

أجسام الظلال (أي الأجسام التي نستخدمها كظلال) في واجهة الظلال Direct3DRMShadow ننشؤها من خلال الأمر Direct3DRM CreateShadow() وكذلك من خلال هذه الواجهة نستطيع أن نستخدم الأمر Init() لإنشاء تلك الأجسام ولكن استخدام هذا الأمر الأخير ليس ضروري لأن الأمر الأول يقوم بعمل الأمر الثاني وبشكل اسهل.

واجهة الحركة DIRECT3DRMANIMATION

عندما نتكلم عن الحركة علينا ملاحظة أنها تحمل معنيان المعنى الأول هو حركة الأجسام في المشهد المعنى الثاني هو الواجهة التي تقوم بإنشاء و تغيير مفاتيح الحركة في المشهد (هذه الواجهة). والحركة هي نتاج تغير موقع وشكل أي جزء من الشاشة (هذا القانون ينطبق على الحركة في الأفلام كذلك).

إنشاء المفاتيح

تسمح هذه الواجهة بإنشاء مفاتيح الحركة. وهذه المفاتيح عبارة عن إطارات (يختلف معنى الإطارات في الحركة عن المعنى الذي سبق وتحدثنا عنه ، هنا تعني الصور المطلوبة لإنشاء الحركة وكل صورة تسمى إطار أو Frame) تحتوي على تغير في شكل الحركة لأي جسم وتقوم Direct3DRM بإنتاج الحركة لكل جسم على

الشاشة بشكل خطي متواصل بمعنى لو أن كرة مثلاً بدأت الحركة بشكل مستقيم في مشهد فإنها سوف تستمر بحركتها المستقيمة إلى أن ينتهي المشهد ما لم نستخدم مفتاح للإطارات (Key Frame) لتغيير تلك الحركة. وباستخدام المفاتيح نستطيع تغيير موقع ووجهة الدوران و حجم أي جسم على الشاشة بحيث نقول لهذه الواجهة بأن الجسم (أ) يجب أن يغير الإطار رقم (20) مثلاً إلى حجم أو موقع أو دوران (وجهة الدوران) معين ويسمى الإطار رقم (20) بمفتاح الإطارات ويمكننا استخدام أي عدد من المفاتيح في حركة معينة.

وتقوم هذه الواجهة في التحكم في مفاتيح الإطارات عن طريق الأوامر التالية:

- أضف مفتاح الموقع `AddPositionKey()`
- أضف مفتاح الدوران `AddRotateKey()`
- أضف مفتاح الحجم `AddScaleKey()`
- امسح مفتاح الإطارات `DeleteKey()`

وكل أمر من هذه الأوامر يتطلب معرفة الوقت و الشكل لكي يقوم بالتغيير ، فالوقت هو النقطة التي يبدأ فيها المفتاح بأجراء التغيير والشكل هو موقع ودرجة دوران و حجم جسم معين والقيمة المرسله للمتغير `SetTime()` يعتمد على القيم التي نعطيها للمفتاح عند إنشائه. وقيم الوقت هي عبارة عن متغيرات من نوع Floating point بحيث نستطيع استخدام مفاتيح الحركة في موقع من الحركة حتى لو كان مجموع الحركة يساوي 1 ، وهذا أيضا يعطينا القدرة في التحكم على سرعة الحركة سواء جعلها سريعة من خلال استخدام إضافات كبيرة أو بطيئة من خلال استخدام إضافات صغيرة عن طريق الأمر `SetTime()` ولذلك فإننا نستطيع عن طريق هذا الأمر أن نوقف ، نبدأ ، نسرع ، نخفض أو نعكس الحركة لأي جسم في المشهد.

وأخيرا يجب علينا معرفة الأمر DeleteKey() والذي يقوم بإزالة أي مفتاح للإطارات غير مرغوب به.

التحكم في الوقت للحركة

عندما ننتهي من إنشاء الحركة نستخدم الأمر SetTime() لمعرفة الموقع الحالي في الحركة. القيم المرسله إلى الأمر SetTime() تعتمد على القيم التي تم استخدامها عند إنشاء مفاتيح الحركة. وكما ذكرنا بأن القيم يجب أن تكون من نوع Floating point حتى يتسنى لنا استخدام مفاتيح الإطارات في أي موقع من الحركة حتى لو كان مجموع الحركة يساوي العدد واحد وهذا أيضا يعطينا القدرة على التحكم في سرعة الحركة (عن طريق القيم المعطاة للأمر SetTime()) نستطيع التحكم في السرعة بحيث تزيد القيم الكبيرة من الحركة وتبطئ القيم الصغيرة منها)

اختيارات الحركة

الأمرين :

- أحصل على الاختيارات GetOptions()
 - ضع الاختيارات SetOptions()
- يستخدمان في التحكم على سلوك الحركة سواء أكان :
- حركة خطية أم متغيرة
 - حركة مفتوحة أم مغلقة
 - حركة تستخدم موقع معين

- حركة تستخدم الحجم ودرجة الدوران

في حالة الحركة الخطية (Linear animation) فإن واجهة الحركة سوف تحرك جسم معين بين مفاتيح الإطارات عن طريق استخدام أقرب مسافة ممكنة بين كل مفتاح وآخر. أم في حالة الحركة المتغيرة (Spline-based animation) تستخدم المنحنيات لحساب موقع الجسم بحيث تكون الحركة أكثر دائرية عند انتقال الجسم بين مفاتيح الإطارات. في حالة الحركة المفتوحة أو المغلقة فإنها تحدد للجسم المتحرك كيفية التعامل مع القيم الخارجة عن مجال الأمر SetTime(). ففي حالة الحركة المغلقة فإن القيم الخارجة عن مجال الأمر SetTime() تضاف إلى بداية الحركة وبذلك فإن الحركة تعاد على قدر تلك الإضافة ، أما في حالة الحركة المغلقة فإن تلك القيم الخارجة عن المجال لا تحتسب لأننا في هذه الحالة نستطيع إعادة الحركة من بدايتها عن طريق إعادة القيمة المعطاة للأمر SetTime().

في حالة استخدام موقع معين فإنها تستخدم مفاتيح الموقع للحركة ، بحيث نتحكم في موقع الجسم عند حركته ونستطيع أن نبطل مفعول هذه الحالة عندما نريد تغيير حجم ودرجة دوران جسم وليس موقعة. وكذلك عند أبطال مفعول هذه الحالة لا يكون لمفاتيح موقع الحركة أي تأثير.

في الحالة الأخيرة وهي استخدام حجم ودرجة دوران معينة لجسم فإننا نتحكم في حجم ودرجة دوران الجسم في مفاتيح الحجم والدوران (كل مفاتيح الحركة هي عبارة عن مفاتيح للإطارات) وعندما تكون هذه الحالة غير مستخدمه فإن مفاتيح الدوران الحجم ليس لها تأثير.

واجهة مجموعة الحركة DIRECT3DRMANIMATIONSET

مجموعة الحركة هي عبارة عن مجموعة أجسام متحركة نستطيع استخدامها لإنتاج مشهد متحرك كامل. العادة عندما نريد أن نستخدم مشهد متحرك فإننا نستخدم أحد البرامج ثلاثية الأبعاد في إنتاج ذلك المشهد ثم نجلبه في برنامجنا ، وكل جسم في مشهد متحرك يُعبّر عنه بجسم متحرك وتشكل هذه الأجسام المتحركة في مجموعها "مجموعة الحركة".

من خلال هذه الواجهة (Direct3DRMAnimationSet) نتحكم في إنتاج مجموعة الحركة والتحكم بها ونستخدم الأمر CreateAmimationSet() في إنشاء مجموعة حركة.

تعبئة (LOADING) مجموعة الحركة

نستطيع استخدام الأمر Load() Direct3DRMAnimationSet في تعبئة مجموعة الحركة من ملف معين ونستخدم اسم الملف المحتوي على مشهد حركة كامل مع الأمر Load() ، ومثل باقي الأوامر المهمة في تعبئة المعلومات في Direct3D فإننا نستطيع أن نعبئ المعلومات من ملفات على القرص الصلب أو من الذاكرة أو من موارد البرنامج. ونستطيع إضافة أجسام متحركة أو إزالتها من مجموعة الحركة عن طريق الأمرين :

- أضف الحركة AddAnimation()
- امسح الحركة DeleteAnimation()

أنواع متغيرات Direct3D (Data Types)

تستخدم Direct3D عدة متغيرات مرات عديدة خلال عمل البرنامج ولأهمية هذه المتغيرات يجب علينا معرفتها حتى يمكننا استغلالها بالشكل الصحيح:

◆ D3DVALUE يعتبر هذا المتغير من أهم متغيرات Direct3D وهو من نوع flot حيث تستخدمه في التعبير عن موقع نقاط الأجسام (vertex coordinates) و شدة الإضاءة (light intensities) وسرعة الدوران (rotation speed) وهكذا. وفي بيئة عمل مثل Visual C++ فإننا نتعامل مع الأرقام على أساس أنها 32 بت (32 bit) ولذلك فإن بيئة العمل سوف تفترض أن كل رقم لم يبرمج على أساس نوع معين أنه من نوع int (هذا إذا لم نستخدم الفاصلة العشرية) أو أنه من نوع double (هذا إذا استخدمنا الفاصلة العشرية) بمعنى آخر انك إذا لم تبرمج الأرقام على أساس أنها من نوع flot فإن بيئة العمل سوف تتعامل معها على أنها مكونة من ثمانية بيتات (8 bytes) بدل أربعة بيتات (لأن النوع flot يستخدم أربعة بيتات) ولهذا السبب ومع معرفتنا أن نوع D3DVALUE اكثر قابلية للتحويل (المقصود هو تحويل البرنامج للعمل على برامج تشغيل أو أجهزة مختلفة) فإنه النوع المفضل للاستخدام في برامج Direct3D وفي حالة انك متأكد بأن برنامجك لن يحول إلى برامج تشغيل أخرى غير windows95 أو windowsNT ولا تريد التعامل مع D3DVALUE في برنامجك تستطيع فعل ذلك عن طريق إضافة التعبير "f" للرقم لتعبر لبيئة العمل نوع الرقم المستخدم ويجب استخدام الفاصلة العشرية مع هذه الطريقة ونرى هنا بعض الأمثلة على ذلك:

```

3dfunction( D3DVALUE( 3 ) ) // طريقة نموذجية وآمنة
3dfunction( D3DVALUE( 3.0 ) ) // طريقة نموذجية وآمنة
3dfunction( 3 ) //          int  تفترض بيئة العمل بأنة متغير
                          بيئة العمل تعطي تحذير -من نوع
3dfunction( 3.0 )//          double  تفترض بيئة العمل بأنة
                          بيئة العمل تعطي تحذير-متغير من نوع
3dfunction( 3.0f ) //          طريقة آمنة ولكن غير قابلة للتحويل
3dfunction( 3f ) //          طريقة غير مسموح بها لأنها يجب
                          استخدام الفاصلة العشرية
    
```

◆ D3DVECTOR هذا كذلك أحد المتغيرات المهمة في Direct3D لأنه يتعامل مع

المتجهات وهو عبارة عن structure نكونها كما يلي:

```

typedef struct _D3DVECTOR (
    union(
        D3DVALUE x;
        D3DVALUE dvX;
    );
    union (
        D3DVALUE y;
        D3DVALUE dY;
    );
    union (
        D3DVALUE z;
        D3DVALUE dvZ;
    );
) D3DVECTOR, *LPD3DVECTOR;
    
```

وكلمة union تستخدم للسماح للمتغيرين x و dvX مثلا بالاستخدام بشكل قابل للتبادل ، ويستخدم هذا المتغير (D3DVECTOR) ليس فقط للتعبير عن المتجهات بل كذلك للتعبير عن النقاط.

◆ D3DCOLOR : تستخدم Direct3D هذا المتغير للتعبير عن الألوان وكل لون هو

عبارة عن خليط ثلاثة ألوان هي "الأحمر ، الأخضر ، الأزرق" بالإضافة إلى ما

يعرف بقناة "ألفا" (عبارة عن لون يعبر عن المناطق الغير مرئية من الصورة) وبتغيير شدة (الشدة تغير من القيمة 0 إلى القيمة 255) هذه المركبات نحصل على لون مختلف.

وهو عبارة عن متغير من نوع DWORD بحيث يستطيع خزن أربع قيم من نوع floating point وتخزن القيم بضرب كل لون بالعدد 255 ثم تحويلها إلى DWORD وهناك أوامر توفرها Direct3D للقيام بهذه العملية.

ونستطيع إعطاء D3DCOLOR الألوان عن طريق الأوامر التالية:

D3DCOLOR color=D3DRGB(1,1,1); // نحصل على اللون الأبيض لأننا
اخترنا 1، 1، 1

أو

D3DCOLOR color=D3DRGBA(1,1,1,0); // نحصل على اللون الأبيض مع قناة
"ألفا" تساوي 0

والنتيجة التي نحصل عليها من هذين الأمرين : D3DRGB() و D3DRGBA() تكون بين صفر و واحد بحيث تقوم D3DCOLOR بعملية الضرب والتحويل لترجمة تلك النتيجة إلى اللون المطلوب ، كذلك لاحظ أننا لا نحتاج إلى استخدام المتغير الأول (D3DVALUE) مع النتيجة التي نحصل عليها لأن D3DCOLOR تقوم بتلك العملية لنا.

ونستطيع عكس العملية بالحصول على الألوان الثلاثة "الأحمر ، الأخضر ، الأزرق" أو الحصول على قناة "ألفا" من لون معين عن طريق الأوامر التالية:

◇ أحصل على اللون الأحمر D3DRMColorGetRed()

◇ أحصل على اللون الأخضر D3DRMColorGetGreen()

◇ أحصل على اللون الأزرق D3DRMColorGetBlue()

◇ أحصل على قناة "ألفا" D3DRMColorGetAlpha()

◆ D3DRMBOX : تستخدم Direct3D هذا المتغير لتتعرف على حجم الجسم وكلا الواجهتين ، واجهة الأجسام (Direct3DRMMesh) و واجهة بنية الأجسام (Direct3DRMMeshBuilder) توفرا لنا الأمر GetBox() بحيث نستخدمه للحصول على أبعاد أي جسم.

ويحتوي المتغير D3DRMBOX على متغيرين من نوع D3DVECTOR على هذا الشكل :

```
typedef struct _D3DRMBOX
{
    D3DVECTOR min, max;
} D3DRMBOX;
```

ومع أننا استخدمنا structure من نوع D3DVECTOR ألا أن المعلومات التي يحتويها هي عبارة عن نقاط وليست متجهات. أما المتغيرين min و max فقد استعملنا لإعطاء الزاويتين المتعاكستين لأصغر صندوق ممكن احتوائه في الجسم المطلوب.

◆ D3DRMVERTEX : في هذا المتغير (وهو من نوع struct) تقوم Direct3D

بشرح النقاط (vertices) في جسم معين وتستخدم كما يلي :

```
typedef struct _D3DRMVERTEX{
    D3DVECTOR position;
    D3DVECTOR normal;
    D3DVALUE tu, tv;
    D3DCOLOR color;
} D3DRMVERTEX;
```

في هذا المتغير نستخدم المتجه position لشرح موقع النقطة (vertex) و نستخدم المتجه normal لشرح اتجاه النقطة (vertex normal) إذا كنا نستخدم Gouraud لجودة النتيجة المرئية أو اتجاه الوجه إذا كنا نستخدم flat لجودة النتيجة المرئية (راجع أنواع الـ Rendering في بداية هذا الفصل لتعرف أكثر عن جودة النتيجة المرئية). أما القيمتين tu و tv يستخدمان لمعرفة الموقع على الشكل الخارجي للجسم لاستخدامه مع النقطة (vertex). أما القيمة color فنستخدمها لإعطاء لون معين للنقطة.

ويقبل هذا المتغير الأمرين:

◇ أحصل على النقطة GetVertices()

◇ ضع النقطة SetVertices()

وهما من أوامر واجهة الأجسام (Direct3DRMMesh) بحيث يستخدم الأمر GetVertices() لتعبئة سلسلة (array) من D3DRMVERTEX بقيم لنقاط تم إدخالها عن طريق الأمر SetVertices(). ويمكننا بمعرفة هذه المعلومة أن نتحكم في شكل أي جسم على الشاشة بحيث نحرك نقاطه (vertex) أو تحريك شكله الخارجي (texture) ويمكننا كذلك أن نغير من متجهات (normals) النقطة أو الوجه عن طريق هذين الأمرين.

◆ D3DRMQUATERNON : عن طريق هذا المتغير نستطيع أن نتحكم في درجة الدوران (rotation) ونحن نعلم أن باستخدام متغير للمتجه وآخر لدرجة الدوران نستطيع أن نتحكم في درجة دوران جسم معين حيث أن المتجه يدل على إحداثي الدوران ودرجة الدوران تدل على المدى. ولكن باستخدام هذا المتغير D3DRMQUATERNON نستطيع أن نستخدم المتجه والدرجة في آن واحد وهو من نوع struct كما يلي:

```
typedef struct _D3DRMQUATERNION {
    D3DVALUE    s;
```

```

D3DVALUE v;
} D3DRMQUATERNION;
typedef D3DRMQUATERNION, *LPD3DRMQUATERNION;

```

ونستخدم الأمر `D3DRMQuaternionFromRotation()` لإنشاء هذا المتغير. ويأخذ هذا الأمر متجهه و درجة دوران لإنشاء `quaternion`. ونستخدم الأمر `D3DRMQuaternionslerp()` لإيجاد متوسط قيم درجات الدوران بحيث يأخذ في الحسبان قيمتين من نوع `quaternion` و قيمة واحدة من نوع `slerp` ثم يقوم بحساب القيمة المتوسطة للقيمتين المعطيتين من نوع `quaternion` بين متجهين أما القيمة من نوع `slerp` فإنها تحدد موقع تلك القيمة المتوسطة بين القيم المعطاة. مثلاً باستخدام `0.5` للقيمة من نوع `slerp` فإن موقع القيمة المتوسطة سوف تكون في الوسط بين القيم المعطاة من نوع `quaternion`.

HRESULT

أغلب أوامر DirectX ترجع لنا قيمة من نوع `HRESULT` في حالة وجود خطأ في تشغيل ذلك الأمر ، و `HRESULT` هو عبارة عن (رقم) قيمة 32 بت (32-bit value) تعطينا حالة ذلك الأمر. و `Direct3D` توفر لنا بعض الثوابت لاستخدامها مع `HRESULT` لكي نعرف طبيعة ذلك الخطأ. في الأحوال العادية أوامر `Direct3DRM` ترجع لنا النتيجة `D3DRM_OK` لتوضح أن الأمر سار كما هو مفترض ، أما إذا حدث خطأ معين فإن القيمة المرجعة (تكون من نوع `HRESULT`) سوف توضح نوع ذلك الخطأ.

ملفات الأكس (X FILES)

تستعمل Direct3D نوعية خاصة من الملفات لحفظ الأجسام ثلاثية الأبعاد يطلق عليها ملفات X وهذه الأجسام تتميز بحسن استخدامها للذاكرة وإعطاء تحكم كامل في ذلك الجسم وكذلك حسن المظهر الخارجي (texture) والألوان والحركة... الخ. ولكن هناك شيئاً واحداً مهم لم يكن من ضمن المجموعة البرمجية لـ Direct3D وهو برنامج إنتاج تلك الأجسام (3D-Editor أو 3D-Software) مما أعطى المبرمجين الفرصة لاستخدام برامجهم الثلاثية الأبعاد المفضلة لإنتاج تلك الأجسام. وطبعاً بدون أداة تمكننا من إنتاج تلك الأجسام لاستخدامها في عوالمنا الثلاثية الأبعاد يتبقى لنا فقط إنتاج أجسام مبدئية بسيطة لا تفي بالغرض المطلوب.

(استعمل النسخة المجانية من البرنامج TrueSpace لإنتاج الأجسام التي تحلو لك. استجده في الملف Extra على القرص المدمج الملحق بالكتاب)

هذه النوعية من الملفات (يطلق عليها كذلك نوعية ملفات DirectX أو DirectX File Format) تعطي المبرمجين نوعية ملفات قادرة على حفظ الكائنات (objects) و أجسام ثلاثية الأبعاد (meshes) والأشكال الخارجية (textures) و مجموعة الحركة (animation sets) بحيث نحصل على قدرة كاملة لتبادل المعلومات بين كل الأجسام المستخدمة في برنامجنا. ومن خلال استغلال قدرة لغة السي في قدرتها على استغلال وتبادل المعلومات بين كل الكائنات فإننا نستطيع من خلال إنشاء كائن واحد لجسم ثلاثي الأبعاد أن نجعله جزءاً من مجموعة أخرى مما يسمح لنا بتخزين تلك المعلومات مرة واحدة واستخدامها بعد ذلك مراراً خلال عمل البرنامج. وتسمح لنا هذه النوعية على تعبئة جسم ثلاثي في برنامجنا ثم حفظ ذلك الجسم فقط أو المجموعة كاملة إذا كان جزءاً من مجموعة (إذا كان ضمن مجموعة مربوط بها فإن الملف الناتج سيحتوي

على المجموعة كلها كجسم واحد) عن طريق الأمر Direct3DRMMeshbuilder Save() و ستكون النوعية للملف الناتج هي نوعية ملفات X. هذه الخواص لهذه النوعية من الملفات هي مجرد بداية إلى جعلها نوعية يمكن تعبئتها وتخزينها من قبل البرامج ثلاثية الأبعاد التي لا تتعامل مع هذه النوعية حتى الآن (وعدت شركة Caligari بأن النسخة الأخيرة من برنامجهم Truespace سوف تقوم بحفظ نوعية ملفات x)، ولحل هذه المشكلة فإنه يأتي من ضمن المجموعة البرمجية لـ DirectX الأمر conv3ds بحيث يأخذ ملف من نوع 3ds (3ds هو نوعية ملفات برنامج 3D-Studio الشهير) ويحولها إلى نوعية x.

الأمر CONV3DS.EXE

باستخدام هذا الأمر نستطيع أن نحول الملفات من نوع 3ds لبرنامج 3D-Studio إلى نوعية x. هناك عدة إضافات مع هذا الأمر تعطينا نتائج مختلفة ونذكر أهمها كما يلي:

m عند إضافة هذا الحرف إلى الأمر تكون النتيجة جسم ثلاثي واحد وليس مجموعة أجسام مركبة

t عند إضافة هذا الحرف إلى الأمر تكون النتيجة ملف x لجسم ثلاثي من نوع (text)

A هذا الأمر يضيف معلومات عن الحركة (إذا كانت الحركة متوفرة) إلى الملف.

f هذا الأمر يعني عدم حفظ معلومات عن الحركة في الملف الناتج

الفصل الخامس عشر

مثال لاستخدام واجهات العنصر

Direct 3D

بعد أن تعرفنا في الفصل السابق على الواجهات والأوامر التي تشكل Direct3D Retained Mode دعونا الآن نتعرف على الخطوات لإنشاء برامج ثلاثية الأبعاد باستخدام Direct3D Retained Mode.

خطوات إنشاء البرنامج :

لكي نستطيع أن ننشئ برنامج Retained Mode سوف نحتاج أولاً للقيام بإنشاء جميع كائنات Direct3D التي سنستخدمها ثم نضيفها إلى المشهد. بعد ذلك سوف نحدد طبيعة الشكل الخارجي للبرنامج (rendering) لكي نستطيع أن نحصل على النتيجة المطلوبة، ثم أخيراً نقوم بإظهار النتيجة النهائية على الشاشة. ونفصل هذه العملية في الخطوات التالية :-

1. نُعرِّف المتغيرات المطلوبة لتخزين القيم المستخدمة لإنشاء برنامج Retained Mode. وتكون هذه المتغيرات عبارة عن مؤشرات (pointers) إلى كائن DirectDraw و كائن Direct3D والمشهد وهكذا.
2. إنشاء النافذة (لأننا في نظام ويندوز فإن كل البرامج لها نتيجة وتستخدم نافذة خاصة بها لإظهارها)
3. بعد إنشاء النافذة سوف نقوم بإنشاء الكائنات التي تحدد المشهد والكائنات المطلوب إرفاقها كجزء من المشهد.
4. كخطوة لإعداد برنامج Retained Mode نقوم أولاً بتحديد نوعية الشكل الخارجي للبرنامج (rendering) وهذه النوعية تحدد شكل الكائنات سواء كان نقاط أو إطار سلكي أو سطحية .. الخ ، بعد أن تتم تلك العملية بنجاح نقوم باختيار "أداة" (device) يعتمد عليها المشهد لإنتاج الشكل الخارجي للبرنامج

- (rendering) واختيار "أداة" مهم لأنه يقرر مثلا استخدام ما على الجهاز من بطاقات تسريع واستغلال إمكانياتها أم لا. وكذلك في هذه الخطوة سوف نقرر على أسس مختلفة كاختيار الإضاءة المناسبة سواء كانت RGB أو ramp وكذلك اختيار السرعات إذا وجدت أو قوة المعالج المستخدم.. الخ.
5. ننشئ الإطار الرئيسي (الإطار الأم) ثم ننشئ إطار للكاميرا. الإطار الأم يحدد لنا أعلى إطار في المشهد وكل الإطارات التابعة (الإطارات الأبناء) توضع تحته بشكل مباشر أو غير مباشر. إطار الكاميرا يتحكم في موقع و اتجاه ودرجة الدوران المطلوب.
6. ثم نضع المشهد بعد أن أنشأناه في الإطار الأم بالشكل المرغوب. وضع الكاميرا في إطارها سوف يحدد المنطقة المرئية في المشهد (viewport).
7. نقوم بإنشاء كائن حاذف (clipping object) بالأبعاد الثنائية (2D) ثم نلحقه بالمنطقة المرئية (viewport) هذا الكائن الحاذف سوف يقوم بحذف الأجزاء الواقعة خارج المنطقة المرئية للكائنات (المنطقة المرئية بالنسبة للمشاهد هي شاشة الكمبيوتر وهي ثنائية الأبعاد) وبذلك تزيد سرعة العرض.
8. ننشئ "الأداة" (device) و المنطقة المرئية (viewport). المنطقة المرئية سوف تحدد المستوى (plane) ثنائي الأبعاد (2D) الذي سيتم عرض المعلومات ثلاثية الأبعاد (3D) عليه.
9. ننشئ الأجسام ثلاثية الأبعاد لأضافتها للمشهد. يجب إضافة الأجسام ثلاثية الأبعاد بعد إنشائها إلى المشهد قبل أن نستطيع استخدامها. وتلك الأجسام يمكن إنشائها من خلال نفس البرنامج (عن طريق برمجة المعلومات عن جسم معين من ضمن البرنامج) أو عن طريق تعيبتها من ملفات موجودة في مسار

البحث (ننشئ هذا النوع من الأجسام عن طريق برامج الرسم ثلاثية الأبعاد و الحقيقة أن هذه هي الطريقة المفضلة لإنتاج برامج ذات مستوى تجاري).

10. ننشئ ونضع الإضاءة في المشهد

11. نقوم بإظهار نافذة البرنامج

12. بعد إظهار نافذة البرنامج يجب أن نضع البرنامج في حلقة (Loop) مستمرة

حتى حدوث حدث. في هذه الخطوة إذا حصل حدث في ويندوز (Windows events) فإن البرنامج يتعامل مع ذلك الحدث عن طريق وظائف البرنامج. إذا لم يوجد أي حدث في ذاكرة الأحداث (event queue) فإن البرنامج يملك الاختيار في أخذ موقف معين أو يقوم بإظهار الإطار التالي في الحركة. أما إذا وجد أي حدث في ذاكرة الأحداث (event queue) فإن البرنامج يقوم بتنفيذه. وللملاحظة فإن ذاكرة الأحداث (event queue) هي عبارة عن جزء من الذاكرة تحتفظ فيه ويندوز بالأحداث التي تستقبلها مثل حركة الفأرة و الضغط على لوحة المفاتيح وتحريك عصا الألعاب... الخ.

◆ المشهد :

بعد أن تعرفنا على الخطوات الأزمة لإنشاء برنامج ثلاثي الأبعاد من نوع Retained Mode وقبل الدخول إلى مثالنا الأول دعونا نتعرف على كيفية إنشاء المشهد ثلاثي الأبعاد في Retained Mode .

المشهد في Retained Mode هو عبارة عن مجموعة إطارات بحيث تقوم Retained Mode بإنشاء الإطار الأم المطلوب لإنشاء سلسلة من الإطار في المشهد. كل الإطارات ماعدا الإطار الأم يملك إطار "أم" وإطار "أم" واحد فقط . وكل إطار قد يكون وقد

لا يكون له إطار أبن تحته. وكل الإطارات لها موقع واتجاه يكون متجانس مع موقع و اتجاه الإطار الأم مما يساعد على ترتيب تسلسل الأجسام في المشهد. بحيث لو غيرنا موقع أو اتجاه أو درجة دوران أي إطار فإن هذا التغيير سوف يطرأ على كل سلسلة الأبناء لذلك الإطار.

كما ذكرنا أن المشهد عبارة عن سلسلة من الإطارات. الإطار الرئيسي أو الإطار "الأم" وهو في أعلى إطارات المشهد وليس له إطار "أم" ثم إطار الكاميرا ويستخدم لتحديد المنطقة المرئية (viewport) وهو ملحق بالإطار الرئيسي ثم نقوم بإضافة إطار العالم "world frame" تحت الإطار الرئيسي حيث نضع فيه الأجسام ثلاثية الأبعاد والإضاءة وذلك حتى يسهل علينا التحكم في الأجسام و الإضاءة في المشهد بدون أن تؤثر على موقع أو حركة أو اتجاه الكاميرا (ذلك لأنها ملققة بإطار خاص بها). ولإضافة أجسام و أضواء إلى المشهد فإنها يجب أما أن تلتحق بإطارات خاصة بها والتي بدورها تلتحق بإطار العالم أو أنها تلتحق بإطار العالم مباشرة. وإذا أردنا إضافة إضاءة طيفية (Ambient light) فإنها يجب أن تلتحق بالإطار الرئيسي.

ولإنشاء المشهد يجب أن نتبع الخطوات التالية :

- ◆ إنشاء الإطار الرئيسي أو الأم
- ◆ إنشاء الكاميرا وإلحاقها بالإطار الرئيسي
- ◆ إنشاء الإضاءة الطيفية وإلحاقها بالإطار الرئيسي
- ◆ إنشاء إطار العالم وإلحاقه بالإطار الرئيسي
- ◆ تعبئة الأجسام وإنشاء الإضاءة وإلحاقهما بإطار العالم

إذا كان ولا بد فإن الأجسام و الإضاءة يمكن أن يكون لها إطاراتها الخاصة بها بحيث تضاف إلى المشهد عن طريق تلك الإطارات. إعطاء إطار لجسم أو ضوء معين يعطينا القدرة على التحكم بذلك الجسم أو الضوء بدون أن نغير أي شيء آخر في المشهد.

إنشاء الإضاءة :

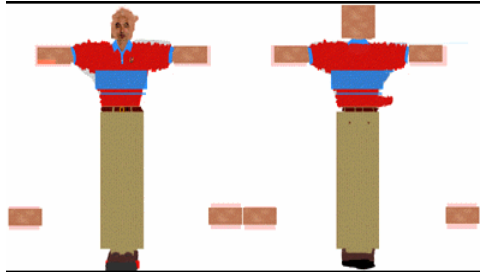
لكي ننشئ إضاءة في المشهد باستخدام إطار خاص بها فإن علينا إتباع الخطوات التالية :

- ◆ ننشئ إطار لتلك الإضاءة
- ◆ ننشئ الإضاءة
- ◆ نلحق الإضاءة بالإطار
- ◆ نضع موقع و اتجاه الإطار مع احترام موقع واتجاه الإطار الأم
- ◆ نلحق إطار الإضاءة بالإطار الأم.

تعبئة الأجسام ثلاثية الإبعاد :

لكي ننشئ أجسام في المشهد باستخدام إطار خاص بها فإن علينا إتباع الخطوات التالية :

- ◆ ننشئ إطار لذلك الجسم
- ◆ ننشئ أو نعبئ الجسم
- ◆ نلحق الجسم بالإطار
- ◆ نضع موقع و اتجاه الإطار مع احترام موقع واتجاه الإطار الأم
- ◆ نلحق إطار الإضاءة بالإطار الأم.



الشكل الخارجي للجسم



الإطار السلكي لجسم ثلاثي الأبعاد



النتيجة النهائية بعد إضافة الشكل الخارجي

مثال لبرنامج Direct3D Retained Mode

في هذا الفصل سوف نرى أمثلة تطبيقية لما سبق وتعلمناه في الفصل السابق وسوف نقوم بإنشاء برنامج ثلاثي الأبعاد باستخدام Direct3D (بعض الأحيان نستخدم المختصر D3D بدل Direct3D) وسوف نستخدم واجهات النوع Retained Mode والتي تعرفنا عليها سابقا. وكما ذكرنا بأن RM (Retained Mode) يوفر لنا سهولة التعامل مع D3D.

وتستخدم D3DRM الكثير من الواجهات التي هي عبارة عن واجهات من نوع COM لتعطينا تحكم كامل لبرمجة محيط ثلاثي الأبعاد. وسوف نبدأ ببرنامج مبسط ليعطينا فكرة عن مدى قدرة RM في إعطاء بيئة العمل (Framework) تلك المميزات.

ما جرت عليه العادة عند البدء في تعلم برمجة لغة من لغات الكمبيوتر كتابة برنامج يقوم بإظهار عبارة "HELLO WORLD" على الشاشة ، لكي يكون هذا البرنامج محدود المستوى (محدود المستوى لأن كل ما يقوم به هو إظهار تلك العبارة) وسريع الفهم قدر المستطاع ولا بد وأن يحتوي على القواعد الرئيسية لتلك اللغة ليتسنى لنا تعلم أساسياتها. وسوف نسير في هذا الفصل على نفس الطريقة حيث سنقوم بالنظر إلى أحد الأمثلة التي تأتي مع D3D بحيث أن كل ما يقوم به البرنامج هو تدوير كرة على المحور الصادي (تدوير كرة وليس إظهار عبارة لأن الهدف هو إنشاء مشهد ثلاثي الأبعاد) والبرنامج بأكمله يكتب في ملف واحد. وبدراسة هذا المثال سوف نتعلم الكثير عن برمجة D3DRM وسوف نتعرف على بيئة Windows وكيفية البرمجة عليها ومع أننا في هذا البرنامج سنتعامل مع محيط

ثلاثي الأبعاد (عن طريق استخدام D3DRM طبعا) إلا أن نفس المنوال ينطبق على كل برامج Windows وعند إنشاء مثالنا سنقوم هنا بالخطوات التالية:

- كيفية بداية المثال وما يقوم به
- إنشاء التعاريف المتغيرات المستخدمة
- إعداد وإنشاء نافذة على الشاشة (سوف نتعرف فيما بعد على كيفية استغلال الشاشة بأكملها)
- إعداد Enumerating Device Drivers
- إعداد محيط ثلاثي الأبعاد
- إنشاء دورة الإخراج (Rendering Loop)
- إنشاء المشهد Creating the Scene
- إنهاء البرنامج Cleaning Up

كيفية بداية المثال وما يقوم به:

في هذا المثال سنقوم بإنشاء برنامج Direct3D Retained-Mode ولتجربة هذا المثال سوف نحتاج إلى Visual-C++5 حيث سنقوم بتعبئة ملف المشروع. هذا المثال نستخدم جسم كرة ثلاثي الأبعاد (Sphere3.X) و صورة مصاحبة للشكل الخارجي (يجب أن تكون أبعاد تلك الصورة من مضاعفات العدد 2 مثلا 16 x 16 أو 1024 x 128... الخ). (يجب ملاحظة أن ملفات الـ headers لـ Direct3D يجب أن تكون موجودة في مسار Include واستخدام المكتبات Winmm.lib و D3drm.lib و Ddraw.lib) هذا المثال هو عبارة عن صورة مبسطة عن المثال Globe والذي يأتي كجزء من المجموعة البرمجية (S.D.K) لـ Direct3D ومثال Globe

مثل بقية أمثلة Direct3DRM في المجموعة البرمجية لـ Direct3D يحتاج إلى إضافة الملف Rmmain.cpp وعدد من ملفات الـ headers. حتى نحافظ على بساطة هذا المثال ولكي يكون سهل الفهم فإن الملف Rmmain.cpp حول من لغة السي المحسنة (C++) إلى لغة السي (C) (الفرق بين اللغتين أحيانا غير واضح للكثير من المبرمجين الجدد لـ C++ و القدامى على C وذلك بسبب التشابه الكبير بين اللغتين، لمعلومات إضافية عن الفروق تستطيع مراجعة الكتب المختصة، ولكن لا حاجة لذلك لفهم هذا المثال) ثم أضيف كجزء من هذا المثال. وحتى نبقيه بسيط الفهم قدر المستطاع فإن كل ما يستطيع المستعمل القيام به هو بدئه أو إنهائه أو تصغيره (minimizing). عند كتابة برامج Direct3DRM فإننا نحتاج إلى الكثير من الأوامر البرمجية المساعدة في إيجاد الأخطاء التي قد تحصل في برامجنا، ولكن لن نستخدم هذه الأوامر في هذا المثال أولا لإبقاء سهولة الفهم ثانيا لعدم الحاجة لذلك (لأنه قد سبق ونقح من جميع الأخطاء المحتملة).

تقريبا كل برامج Direct3D تستخدم DirectDraw لإظهار النتيجة المرئية على الشاشة. وهذا البرامج قد تستخدم DirectDraw للشاشة بكاملها (full-screen) عن طريق exclusive mode أو كنافذة من نوافذ ويندوز. ومثالنا يستخدم نافذة من نوافذ ويندوز لإظهار النتيجة المرئية. ومع أن استخدام الشاشة بكاملها يعطينا بعض المميزات كالسرعة والتحكم في شكل الشاشة بالكامل إلا أن كتابة برنامج يستخدم نافذة من نوافذ ويندوز يعتبر اسهل لتنقيحه من الأخطاء (code debugging). ويقوم الكثير من المبرمجين بكتابة برامجهم كنافذة من نوافذ ويندوز وعند الانتهاء منها يقوم بتحويلها إلى الشاشة بأكملها بعد إزالة الأخطاء البرمجية التي قد تكون موجودة في تلك البرامج.

إنشاء تعاريف المتغيرات المستخدمة:

سوف نبدأ بالأسطر الأولى من البرنامج (لا تنسى أن البرنامج بأكمله في ملف واحد). في البداية سوف نقوم بتعريف المتغيرات المستخدمة، سوف نبدأ بتعريف المتغير INITGUID (يجب ملاحظة أنه عند برمجة DirectX هذا المتغير يجب تعريفه في بداية البرنامج قبل أي أمر آخر):

```
#define INITGUID // defines و includes من يستهل غيرة
#include <windows.h>
#include <malloc.h> // Required by memset call
#include <d3drmwin.h>

#define MAX_DRIVERS 5 // Maximum D3D drivers expected

// Global variables

LPDIRECT3DRM lpD3DRM; // Direct3DRM object
LPDIRECTDRAWCLIPPER lpDDClipper; // DirectDrawClipper object

struct _myglobs {
    LPDIRECT3DRMDEVICE dev; // Direct3DRM device
    LPDIRECT3DRMVIEWPORT view; // Direct3DRM viewport
    through
    LPDIRECT3DRMFRAME scene; // which the scene is viewed
    // Master frame in which
    // others are placed
    LPDIRECT3DRMFRAME camera; // Frame describing the
    user's
    // POV

    GUID DriverGUID[MAX_DRIVERS]; // GUIDs of available
    D3D
    // drivers
    char DriverName[MAX_DRIVERS][50]; // Names of available
    // D3D drivers
    int NumDrivers; // Number of available
    // D3D drivers
    int CurrDriver; // Number of D3D driver
    // currently
    // being used
}
```

```

    BOOL bQuit;                // Program is about to
                               // terminate
    BOOL bInitialized;        // All D3DRM objects are
                               // initialized
    BOOL bMinimized;          // Window is minimized

    int BPP;                   // Bit depth of the current
                               // display mode

} myglobals;

```

نضع بعد ذلك تعاريف الوظائف والمتغيرات المستعملة في البرنامج :

```

// Function prototypes.

static BOOL InitApp(HINSTANCE, int);
long FAR PASCAL WindowProc(HWND, UINT, WPARAM, LPARAM);
static BOOL EnumDrivers(HWND win);
static HRESULT WINAPI enumDeviceFunc(LPGUID lpGuid,
    LPSTR lpDeviceDescription, LPSTR lpDeviceName,
    LPD3DDEVICEDESC lpHWDesc, LPD3DDEVICEDESC lpHELDesc,
    LPVOID lpContext);
static DWORD BPPToDDBD(int bpp);
static BOOL CreateDevAndView(LPDIRECTDRAWCLIPPER lpDDClipper,
    int driver, int width, int height);
static BOOL SetRenderState(void);
static BOOL RenderLoop(void);
static BOOL MyScene(LPDIRECT3DRMDEVICE dev, LPDIRECT3DRMVIEWPORT
view,
    LPDIRECT3DRMFRAME scene, LPDIRECT3DRMFRAME camera);
void MakeMyFrames(LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME
lpCamera,
    LPDIRECT3DRMFRAME * lplpLightFrame1,
    LPDIRECT3DRMFRAME * lplpWorld_frame);
void MakeMyLights(LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME
lpCamera,
    LPDIRECT3DRMFRAME lpLightFrame1,
    LPDIRECT3DRMLIGHT * lplpLight1, LPDIRECT3DRMLIGHT * lplpLight2);
void SetMyPositions(LPDIRECT3DRMFRAME lpScene,
    LPDIRECT3DRMFRAME lpCamera, LPDIRECT3DRMFRAME lpLightFrame1,
    LPDIRECT3DRMFRAME lpWorld_frame);

```

```
void MakeMyMesh(LPDIRECT3DRMMESHBUILDER * lpSphere3_builder);
void MakeMyWrap(LPDIRECT3DRMMESHBUILDER sphere3_builder,
               LPDIRECT3DRMWRAP * lpWrap);
void AddMyTexture(LPDIRECT3DRMMESHBUILDER lpSphere3_builder,
                 LPDIRECT3DRMTEXTURE * lpTex);
static void CleanUp(void);
```

إعداد وإنشاء نافذة على الشاشة:

بعدما رأينا كيفية استخدام التعاريف و المتغيرات في مثالنا سوف نلقي نظرة هنا على الجزء الذي يليه وهو الجزء المهم بإنشاء وعداد برنامج ويندوز (وهو مشابه لكل برامج ويندوز المستخدمة للنوافذ) وينقسم هذا الجزء إلى ثلاثة أقسام :

1. وظيفة WinMain (WinMain Function)

2. وظيفة InitApp (InitApp Function)

3. عملية إنشاء النافذة الرئيسية (Main Window Procedure)

◀ وظيفة WinMain (WinMain Function) :

يحتوي جزء WinMain في برنامجنا على "بعض" الأوامر الخاصة ببرامج DirectDraw و Direct3DRM. والوظيفتين InitApp و CleanUp تستخدمان في كل برامج ويندوز بشكل عام ، وفي حالتنا هذه فإنهما يقومان ببعض الوظائف الخاصة بـ DirectX. وأهم استدعاء (Function Call) تقوم به WinMain لكي تأخذ في الاعتبار Direct3D هو استدعاء وظيفة RenderLoop وهي المسئولة عن تجديد الحركة على الشاشة (معلومات إضافية عن RenderLoop راجع القسم أو الجزء الخاص بها في هذا البرنامج):

```
int PASCAL
```

```

WinMain (HINSTANCE this_inst, HINSTANCE prev_inst, LPSTR
cmdline,
        int cmdshow)
{
    MSG      msg;
    HACCEL  accel = NULL;
    int      failcount = 0; // Number of times RenderLoop has
                          // failed

    prev_inst;
    cmdline;

    // Create the window and initialize all objects needed to
    // begin rendering.

    if (!InitApp(this_inst, cmdshow))
        return 1;

    while (!myglobs.bQuit) {

        // Monitor the message queue until there are no
        // pressing messages.

        while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
            if (!TranslateAccelerator(msg.hwnd, accel, &msg))
            {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
        }

        // If the app is not minimized, not about to quit,
and
        // D3DRM has been initialized, begin rendering.

        if (!myglobs.bMinimized && !myglobs.bQuit &&
            myglobs.bInitialized) {

            // Attempt to render a frame. If rendering fails
more than
            // twice, abort execution.

            if (!RenderLoop())
                ++failcount;
            if (failcount > 2) {
                CleanUp();
                break;
            }
        }
    }
}

```

```

}
return msg.wParam;
}

```

◀ وظيفة InitApp (InitApp Function):

في هذا الجزء سنقوم بإعداد البرنامج وإنشاء كائن (Class Object) لنافذة ثم إنشاء النافذة نفسها للبرنامج كما هو الحال مع جميع برامج ويندوز ثم بعد ذلك تقوم وظيفة **InitApp** بإضافة بعض الأوامر الخاصة بـ **DirectDraw** و **Direct3D**. تقوم وظيفة **IniApp** بالحصول على البت لكل نقطة في الشاشة (bits per pixel) وهذه المعلومات تستخدم لإعطاء النوعية (Quality) لـ **Rendering**. وتقوم بعد ذلك باستدعاء الأمر **EnumDrivers** بشكل محلي (locally defined) لمعرفة نوعية البطاقات الموجودة على الجهاز والتي يمكن لـ **Direct3D** استغلالها (كل بطاقة تتركب على الكمبيوتر تأتي مع برنامج يجعلها تتعامل مع نظام التشغيل ويسمى ذلك البرنامج "أداة" أو **Driver** وكل الشركات تصمم بطاقتها الآن لاستغلال قوة **DirectX** وتعرف إمكانيات تلك البطاقة لـ **DirectX** عن طريق تلك الأداة) ثم تقوم باختيار إحداها عند وجود أكثر من بطاقة. بعد ذلك تقوم باستدعاء الأمر **Direct3DRMCreate** لإنشاء واجهة **IDirect3DRM** (كما شرحنا سابقاً أن هذا أول ما يجب فعله لاستخدام واجهات **Direct3DRM**) حيث تقوم بدورها باستدعاء أمر لإنشاء إطار "أم" (عن طريق إنشاء واجهة الإطار **IDirect3DRM::CreateFrame** وأخر لإعطاء ذلك الإطار موقع معين يستخدم للكاميرا في المشهد **IDirect3DRMFrame::SetPosition**).

وأخيراً عن طريق إنشاء كائن من نوع `DirectDrawClipper` عن طريق الأمر `DirectDrawCreateClipper` ويقوم هذا الأمر الأخير بإنشاء واجهة `Clipper` (هذه الواجهة من واجهات `DirectDraw` وليست من واجهات `Direct3DRM`) نستطيع أن نتحكم من خلالها في إطار الشاشة وفي المناطق الظاهرة عليها والغير ظاهرة ثم نستخدم الأمر `IDirectDrawCipper::SetHWND` لوضع "Handle" للبرنامج وهو عبارة عن متغير يحمل رقم معين (تقوم ويندوز بإعطاء البرنامج هذا الرقم وكل برنامج له رقم خاص به) تتعرف ويندوز على البرنامج من خلاله ، لمعلومات إضافية عن هذا المتغير راجع أحد كتب برمجة ويندوز.

بعد ذلك تقوم وظيفة `InitApp` باستدعاء المتغير المحلي (locally defined) `CreatDevAndView` لإنشاء كائن ووجه النظر لـ `Direct3D`. لمعلومات إضافية راجع إنشاء كائن ووجه نظر.

بعد ذلك يكون البرنامج جاهز لاستقبال أوامر `Direct3DRM` ونستطيع البدء في إنشاء المشهد الثلاثي الأبعاد 3-D سوف ننشئ وظيفة نطلق عليها الاسم `MyScene` من خلالها نقوم بإنشاء ذلك المشهد. وأخيراً تقوم وظيفة `InitApp` بإظهار وتجديد نافذة برنامجنا.

```
static BOOL
InitApp(HINSTANCE this_inst, int cmdshow)
{
    HWND win;
    HDC hdc;
    WNDCLASS wc;
    RECT rc;

    // Set up and register the window class.

    wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc = WindowProc;
    wc.cbClsExtra = 0;
```

```

wc.cbWndExtra = sizeof(DWORD);
wc.hInstance = this inst;
wc.hIcon = LoadIcon(this inst, "AppIcon");
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
wc.lpszMenuName = NULL;
wc.lpszClassName = "D3DRM Example";
if (!RegisterClass(&wc))
    return FALSE;

// Initialize the global variables.

memset(&myglobs, 0, sizeof(myglobs));

// Create the window.

win =
    CreateWindow
    (
        "D3DRM Example",           // Class
        "Hello World (Direct3DRM)", // Title bar
        WS_VISIBLE | WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU
        |
        WS_MINIMIZEBOX | WS_MAXIMIZEBOX,
        CW_USEDEFAULT,             // Init. x pos
        CW_USEDEFAULT,             // Init. y pos
        300,                        // Init. x size
        300,                        // Init. y size
        NULL,                       // Parent window
        NULL,                       // Menu handle
        this inst,                  // Program handle
        NULL                        // Create parms
    );
if (!win)
    return FALSE;
// Record the current display bits-per-pixel.

hdc = GetDC(win);
myglobs.BPP = GetDeviceCaps(hdc, BITSPIXEL);
ReleaseDC(win, hdc);

// Enumerate the D3D drivers and select one.

if (!EnumDrivers(win))
    return FALSE;

// Create the D3DRM object and the D3DRM window object.

lpD3DRM = NULL;
Direct3DRMCreate(&lpD3DRM);

```

```
// Create the master scene frame and camera frame.

lpD3DRM->lpVtbl->CreateFrame(lpD3DRM, NULL,
                            &myglobs.scene);
lpD3DRM->lpVtbl->CreateFrame(lpD3DRM, myglobs.scene,
                            &myglobs.camera);
myglobs.camera->lpVtbl->SetPosition(myglobs.camera,
myglobs.scene, D3DVAL(0.0), D3DVAL(0.0), D3DVAL(0.0));

// Create a DirectDrawClipper object and associate the
// window with it.

DirectDrawCreateClipper(0, &lpDDClipper, NULL);
lpDDClipper->lpVtbl->SetHWND(lpDDClipper, 0, win);

// Create the D3DRM device by using the selected D3D
// driver.

GetClientRect(win, &rc);
if (!CreateDevAndView(lpDDClipper, myglobs.CurrDriver,
rc.right,rc.bottom)) {
    return FALSE;
}

// Create the scene to be rendered.

if (!MyScene(myglobs.dev, myglobs.view, myglobs.scene,
            myglobs.camera))
    return FALSE;

myglobs.bInitialized = TRUE; // Initialization completed

// Display the window.

ShowWindow(win, cmdshow);
UpdateWindow(win);

return TRUE;
}
```


عملية إنشاء النافذة الرئيسية—Main Window Procedure

: (Windowproc)

عملية إنشاء النافذة الرئيسية في هذا البرنامج بسيطة لأنه لا يوجد أي تعامل مع المستعمل فكل ما يقوم به البرنامج هو إظهار كرة تدور حول نفسها. لذلك عندما تستقبل Windowproc رسالة الأمر WM_DESTROY تقوم باستدعاء وظيفة إنهاء البرنامج (Cleaning Up). وعندما تستقبل رسالة الأمر WM_ACTIVATE تقوم باستدعاء الواجهة البديلة IDirect3DRMWinDevice ثم تقوم باستدعاء الأمر HandleActivate وهو أحد أوامر تلك الواجهة وذلك للتأكد بأن الألوان المستخدمة في نافذة برنامجنا هي الألوان المطلوبة ويقوم هذا الأمر بدورة بالتعامل مع رسالة الأمر WM_PAINT عن طريق استدعاء أمر آخر من أوامر الواجهة البديلة وهو

.HandlePaint

```
LONG FAR PASCAL WindowProc(HWND win, UINT msg,
    WPARAM wparam, LPARAM lparam)
{
    RECT r;
    PAINTSTRUCT ps;
    LPDIRECT3DRMWINDEVICE lpD3DRMWinDev;

    switch (msg)    {

    case WM_DESTROY:
        CleanUp();
        break;

    case WM_ACTIVATE:
        {

            // Create a Windows-specific D3DRM window device to
            handle this
            // message.

            LPDIRECT3DRMWINDEVICE lpD3DRMWinDev;
            if (!myglobs.dev)
```

```
        break;
        myglobs.dev->lpVtbl->QueryInterface(myglobs.dev,
&IID IDirect3DRMWinDevice, (void **) &lpD3DRMWinDev);
        lpD3DRMWinDev->lpVtbl->HandleActivate(lpD3DRMWinDev,
(WORD) wparam);
        lpD3DRMWinDev->lpVtbl->Release(lpD3DRMWinDev);
    }
    break;

case WM_PAINT:
    if (!myglobs.bInitialized || !myglobs.dev)
        return DefWindowProc(win, msg, wparam, lparam);

    // Create a Windows-specific D3DRM window device to
    // handle this
    // message.

    if (GetUpdateRect(win, &r, FALSE)) {
        BeginPaint(win, &ps);
        myglobs.dev->lpVtbl->QueryInterface(myglobs.dev,
&IID IDirect3DRMWinDevice, (void **)
&lpD3DRMWinDev);
        if (FAILED(lpD3DRMWinDev->lpVtbl->HandlePaint
(lpD3DRMWinDev, ps.hdc)))
            lpD3DRMWinDev->lpVtbl->Release(lpD3DRMWinDev);
        EndPaint(win, &ps);
    }
    break;
default:
    return DefWindowProc(win, msg, wparam, lparam);
}
return 0L;
```

إعداد الأدوات Enumerating Device Drivers :

تقوم برامج Direct3D باستدعاء أداة (drivers) البطاقات المتوفرة على الجهاز وتختار أفضل واحدة اعتمادا على احتياجات البرنامج. وسوف نقسم هذه العملية في

برنامجنا إلى ثلاث أقسام :

1. الأمر EnumDrivers

2. الأمر enumDeviceFunc

3. الأمر المساعد BPPToDDBD

◀ الأمر EnumDrivers (أمر الإعداد) :

يطلب هذا الأمر عن طريق الوظيفة InitApp قبيل إنشاء المشهد والكاميرا ، ولنتعرف على هذا الأمر جيدا يجب علينا معرفة أن واجه الـ COM هي في الحقيقة عبارة عن واجه لكائن DirectDraw لذلك فإن أول ما يقوم به أمر الإعداد هذا هو إنشاء كائن DirectDraw عن طريق الأمر DirectDrawCreate ثم يقوم باستخدام الأمر QueryInterface (ملاحظة مهمة : استخدام الأمر QueryInterface هو الأسلوب المفضل لإنشاء كائن من كائن آخر في DirectX) بإنشاء واجهة IDirect3D. كذلك لاحظ أننا عند استخدام الأمر QueryInterface في لغة السي استخدمنا عنوان الواجهة كمدخل ثاني لهذا الأمر وليس مجرد ثابت كما هو الحال لو استخدمنا لغة السي المحسنة(راجع هذه الملاحظة مرة أخرى) .

أمر الإعداد تقوم به واجهة Direct3D عن طريق الأمر EnumDevices التي بطريقتها تعتمد على الوظيفة المحلية (locally defined) enumDeviceFunc. ولاحظ أن أمر الإعداد تقوم به واجهة Direct3D وليست واجهة Direct3DRM لعدم وجود أمر

إعداد في واجهة Direct3DRM، ولذلك نستفيد في كيفية استخدام أمر واحد في كلا النوعين من أنواع Direct3D الـ Direct3DRM و Direct3DIM.

```
static BOOL
EnumDrivers(HWND win)
{
    LPDIRECTDRAW lpDD;
    LPDIRECT3D lpD3D;
    HRESULT rval;

    // Create a DirectDraw object and query for the Direct3D
    // interface
    // to use to enumerate the drivers.

    DirectDrawCreate(NULL, &lpDD, NULL);
    rval = lpDD->lpVtbl->QueryInterface(lpDD, &IID_IDirect3D,
        (void**) &lpD3D);
    if (rval != DD_OK) {
        lpDD->lpVtbl->Release(lpDD);
        return FALSE;
    }

    // Enumerate the drivers, setting CurrDriver to -1 to
    // initialize the
    // driver selection code in enumDeviceFunc.

    myglobs.CurrDriver = -1;
    lpD3D->lpVtbl->EnumDevices(lpD3D, enumDeviceFunc,
        &myglobs.CurrDriver);

    // Ensure at least one valid driver was found.

    if (myglobs.NumDrivers == 0) {
        return FALSE;
    }
    lpD3D->lpVtbl->Release(lpD3D);
    lpDD->lpVtbl->Release(lpDD);

    return TRUE;
}
```

◀ الأمر enumDeviceFunc :

هذا الأمر ينتمي إلى نوعية من الأوامر يطلق عليها **D3DENUMDEVICESCALLBACK** ومعرفة في ملف التعريف **D3dcaps.h** ويقوم نظام التشغيل بطلب هذا الأمر وإعطاءه أسم و رقم تعريف (identifiers) لكل سواقة (driver) مستعملة لـ **Direct3D** كذلك إعطاءه معلومات عن قدرة الجهاز سواء كانت تلك القدرة عبارة عن قوة المعالج فقط أو قوة المعالج و بطاقات التسريع المتوفرة معه.

ويستخدم هذا الأمر أمر آخر وهو **dcmColorModel** وهو أحد أوامر **D3DDEVICEDESC** لاختبار قدرة الجهاز ، فإذا وجدت معلومات عن وجود سرعات في الجهاز فإن هذا الأمر يقوم باختبار تلك المعلومات.

بعد ذلك يقوم هذا الأمر بتقرير إذا كانت سواقة الأداة (device driver) قادرة على إعداد الجهاز لإعطاء النتيجة المطلوبة للبرنامج وإعطاء عدد الألوان المطلوب أم لا. وعلية إذا كانت سواقة الأداة لا تستطيع فإن الأمر يرجع **D3DENUMRET_OK** وذلك ليتجنب تكملة إعداد سواقة الأداة التالية (في حالة وجود أكثر من أداة للجهاز). ويستخدم هذا الأمر كذلك الأمر **BPPToDDBD** (هذا الأمر يعني البت لكل نقطة على الشاشة لعدد الألوان لـ **DirectDraw**، راجع القسم الثالث "الأمر المساعد **BPPToDDBD**" لترى كيفية استعمال هذا الأمر في البرنامج) لمقارنة المعلومات الموجودة عن عدد الألوان مع البت لكل نقطة على الشاشة (-bits-per-pixel) والذي حصلنا عليه عن طريق الأمر **GetDeviceCaps** الموجود في وظيفة **.InitApp**.

فإذا استطاعت سواقة الأداة تحت الإعداد النجاح في هذه الاختبارات البسيطة يتم استعمال بقية أوامر الاختبار لـ D3DDEVICEDESC وعلية يتم اختيار ما على الجهاز من سرعات (hardware acceleration) بدل قوة المعالج فقط وكذلك سيتم اختيار النوع RGB للإضاءة بدل النوع .ramp.

```

////////////////////////////////////
//
// enumDeviceFunc
// Callback function that records each usable D3D driver's name
// and GUID. Chooses a driver and sets *lpContext to this driver.
//
////////////////////////////////////

static HRESULT
WINAPI enumDeviceFunc(LPGUID lpGuid, LPSTR lpDeviceDescription,
    LPSTR lpDeviceName, LPD3DDEVICEDESC lpHWDesc,
    LPD3DDEVICEDESC lpHELDesc, LPVOID lpContext)
{
    static BOOL hardware = FALSE; // Current start driver is hardware
    static BOOL mono = FALSE;     // Current start driver is mono
    light
    LPD3DDEVICEDESC lpDesc;
    int *lpStartDriver = (int *)lpContext;

    // Decide which device description should be consulted.

    lpDesc = lpHWDesc->dcmColorModel ? lpHWDesc : lpHELDesc;

    // If this driver cannot render in the current display bit-depth,
    // skip it and continue with the enumeration.

    if (!(lpDesc->dwDeviceRenderBitDepth & BPPToDDBD(myglobs.BPP))
        return D3DENUMRET_OK;

    // Record this driver's name and GUID.

    memcpy(&myglobs.DriverGUID[myglobs.NumDrivers], lpGuid,
        sizeof(GUID));
    lstrcpy(&myglobs.DriverName[myglobs.NumDrivers][0],
lpDeviceName);

    // Choose hardware over software, RGB lights over mono lights.

```

```

if (*lpStartDriver == -1) {

    // This is the first valid driver.

    *lpStartDriver = myglobs.NumDrivers;
    hardware = lpDesc == lpHWDesc ? TRUE : FALSE;
    mono = lpDesc->dcmColorModel & D3DCOLOR_MONO ? TRUE : FALSE;
} else if (lpDesc == lpHWDesc && !hardware) {

    // This driver is hardware and the start driver is not.

    *lpStartDriver = myglobs.NumDrivers;
    hardware = lpDesc == lpHWDesc ? TRUE : FALSE;
    mono = lpDesc->dcmColorModel & D3DCOLOR_MONO ? TRUE : FALSE;
} else if ((lpDesc == lpHWDesc && hardware) ||
            (lpDesc == lpHELDesc && !hardware)) {
    if (lpDesc->dcmColorModel == D3DCOLOR_MONO && !mono) {

        // This driver and the start driver are the same type,
        // and
        // this driver is mono whereas the start driver is not.

        *lpStartDriver = myglobs.NumDrivers;
        hardware = lpDesc == lpHWDesc ? TRUE : FALSE;
        mono = lpDesc->dcmColorModel & D3DCOLOR_MONO ? TRUE :
        FALSE;
    }
}
myglobs.NumDrivers++;
if (myglobs.NumDrivers == MAX_DRIVERS)
    return (D3DENURET_CANCEL);
return (D3DENURET_OK);
}

```

◀ الأمر المساعد BPPToDDBD :

يستعمل هذا لتحويل البت لكل نقطة على الشاشة للأداة (device) المستعملة حالياً من قبل نظام التشغيل إلى رقم نستطيع مقارنته مع عدد الألوان لسواقة الأداة التي يتم إعدادها.

```

////////////////////////////////////
//
// BPPToDDBD
// Converts bits-per-pixel to a DirectDraw bit-depth flag.
//
////////////////////////////////////

static DWORD
BPPToDDBD(int bpp)
{
    switch(bpp) {
        case 1:
            return DDBD_1;
        case 2:
            return DDBD_2;
        case 4:
            return DDBD_4;
        case 8:
            return DDBD_8;
        case 16:
            return DDBD_16;
        case 24:
            return DDBD_24;
        case 32:
            return DDBD_32;
        default:
            return 0;
    }
}

```

إعداد محيط ثلاثي الأبعاد :

في هذا القسم من البرنامج سوف نقوم بشرح الخطوات المطلوبة لإنشاء محيط أو عالم ثلاثي الأبعاد (3-D) ونقوم بذلك في خطوتين:

1. إنشاء الأداة ووجهة النظر

2. وضع مستوى الشكل الخارجي للبرنامج (Render State)

لا تقوم هذه الوظيفة بوضع الأجسام أو الإطارات أو الإضاءة في المحيط الثلاثي الأبعاد لأن هذه وظيفة MyScene وأوامرها ويمكنك مراجعة تلك الوظيفة في قسم "إنشاء المشهد".

إنشاء الأداة ووجهة النظر : (نعم تحتاج لقراءته أكثر من مرة حتى تستطيع فهمه)

أن الأداة ووجهة النظر التي سوف تستخدمها Direct3D تُنشأ عند بداية البرنامج (application's initialization) بعد إنشاء كائن الحذف لـ DirectDraw (DirectDrawClipper Object) حيث تقوم وظيفة InitApp باستدعاء الأمر CreateDevAndView (لاحظ أن الأمر يتكون من أربع كلمات الأولى Create وتعني "أنشئ" والثانية Dev وهي اختصار كلمة Device وتعني "أداة" والثالثة And وهي حرف العطف "و" ثم الأخيرة View وتعني "وجهة نظر" ، أي أن الأمر يقول "أنشئ الأداة ووجه النظر". وهذه القاعدة تنطبق على كل الأوامر تقريبا) الذي بدوره يطلب الكائن الحاذف والأداة المختارة وأبعاد الشاشة المرئية.

ولإنشاء أداة لـ Direct3DRM يستخدم الأمر CreateDevAndView أحد أوامر Direct3DRM وهو IDirect3DRM::CreatDeviceFromClipper ويعني أنشئ أداة (أو واجهة في هذه الحالة-راجع الواجهة الرئيسية في الفصل السابق) من الكائن الحاذف. وتستخدم الأداة التي تم اختيارها سابقاً عن طريق أوامر الإعداد. ويستخدم الأمر CreateDevAndView كذلك بعض أوامر الواجهة البديلة وهي GetWidth() للحصول على عرض الشاشة (عرض الشاشة المستخدمة من قبل الأداة الجديدة) وكذلك الأمر GetHeight() للحصول على ارتفاع الشاشة (ارتفاع الشاشة

المستخدمة من قبل الأداة الجديدة) بعد الحصول على المعلومات المطلوبة يقوم باستخدام الأمر `CreateViewport` وهو أحد أوامر الواجهة الرئيسية وذلك لإنشاء واجهة وجهة النظر.

وبعدما يقوم الأمر `CreateDevAndView` باستدعاء الأمر `SetBack()` وهو أحد أوامر واجهة وجهة النظر وذلك لوضع الخلفية المحذوفة لواجهة النظر والتي بدورها تستدعي الأمر المحلي `(locally defined) SetRenderState()` وهذا الأمر مشروح في الجزء التالي "وضع مستوى الشكل الخارجي للبرنامج".

```

////////////////////////////////////
//
// CreateDevAndView
// Create the D3DRM device and viewport with the given D3D driver and
// with the specified size.
//
////////////////////////////////////

static BOOL
CreateDevAndView(LPDIRECTDRAWCLIPPER lpDDClipper, int driver,
                 int width, int height)
{
    HRESULT rval;

    // Create the D3DRM device from this window by using the
    // specified
    // D3D driver.

    lpD3DRM->lpVtbl->CreateDeviceFromClipper(lpD3DRM, lpDDClipper,
        &myglobs.DriverGUID[driver], width, height, &myglobs.dev);

    // Create the D3DRM viewport by using the camera frame. Set the
    // background depth to a large number. The width and height
    // might have been slightly adjusted, so get them from the
    // device.

    width = myglobs.dev->lpVtbl->GetWidth(myglobs.dev);
    height = myglobs.dev->lpVtbl->GetHeight(myglobs.dev);
    rval = lpD3DRM->lpVtbl->CreateViewport(lpD3DRM, myglobs.dev,
        myglobs.camera, 0, 0, width, height, &myglobs.view);
    if (rval != D3DRM_OK) {
        myglobs.dev->lpVtbl->Release(myglobs.dev);
    }
}

```

```

    return FALSE;
}
rval = myglobs.view->lpVtbl->SetBack(myglobs.view,
    D3DVAL(5000.0));
if (rval != D3DRM_OK) {
    myglobs.dev->lpVtbl->Release(myglobs.dev);
    myglobs.view->lpVtbl->Release(myglobs.view);
    return FALSE;
}

// Set the render quality, fill mode, lighting state,
// and color shade info.

if (!SetRenderState())
    return FALSE;
return TRUE;
}

```

وضع مستوى الشكل الخارجي للبرنامج (Render State):

عندما نستخدم Direct3D لإنشاء برنامج معين فإن علينا وضع مستويات معينة لبعض الأوامر مثل مستوى الإضاءة ومستوى الشكل الخارجي و مستوى حركة (transformation) الأجسام ونقوم بذلك عن طريق استخدام أوامر Retained-Mode واستخدام تلك الأوامر بسيط مقارنة باستخدام النوع الآخر من أنواع Direct3D وهو Immediate Mode حيث أننا في النوع الأول لا نرى الكثير من العمل الذي يجب القيام به في حالة استخدام النوع الثاني.

نقوم أولاً باستخدام الأمر IDirect3dRMDevice::SetQuality عن طريق الوظيفة (function) SetRenderState وذلك بتشغيل الإضاءة ووضع الأمر fill (عبئ) لـ solid والجودة لـ Gouraud بعد ذلك إذا أردنا تغيير الاهتياج (dithering) فإننا نستطيع استخدام الأمر IDirect3dRMDevice::SetDither أو تغيير جودة خريطة الشكل الخارجي (texture quality) عن طريق الأمر (اكمل في الصفحة التالية)

IDirect3dRMDevice::SetTextureQuality كما يلي:

```

////////////////////////////////////
////////
//
// SetRenderState
// Set the render quality and shade information.
//
////////////////////////////////////
////////

BOOL
SetRenderState(void)
{
    HRESULT rval;

    // Set the render quality (light toggle, fill mode, shade
    // mode).

    rval = myglobs.dev->lpVtbl->SetQuality(myglobs.dev,
        D3DRMLIGHT_ON | D3DRMFILL_SOLID |
D3DRMSHADE_GOURAUD);
    if (rval != D3DRM_OK) {
        return FALSE;
    }

    // If you want to change the dithering mode, call
    // SetDither here.

    // If you want a texture quality other than
    // D3DRMTEXTURE_NEAREST
    // (the default value), call SetTextureQuality here.

    return TRUE;
}

```

إنشاء دورة الإخراج (Rendering Loop) :

في هذا القسم سوف ننظر إلى وظيفة Rendering Loop وهذه الوظيفة تستدعي عن طريق الوظيفة الرئيسية WinMain (WinMain تشابهه Main في برامج Dos) وتقوم برسم و تجديد كل صورة و إطار جديد للحركة على الشاشة، مع أهمية Rendering Loop إلا أنها تقوم ببعض المهام البسيطة انفصلها كما يلي:

1. تقوم باستدعاء أمر الحركة `IDirect3DRMFrame::Move` للتحكم في درجة دوران (rotations) و تسارع (velocities) الإطارات في سلسلة المشهد.
2. تقوم باستدعاء أمر مسح الشاشة `IDirect3DRMViewport::Clear` ويقوم بمسح خلفية وجهة النظر و تبديلها بالون الخلفي المستخدم.
3. تقوم باستدعاء أمر إخراج الصورة الجديدة على الشاشة عن طريق الأمر `IDirect3DRMViewport::Render` وذلك لإخراج الصورة الجديدة للمشهد الحالي في وجهة النظر.
4. وأخيرا تقوم باستدعاء أمر التجديد `IDirect3DRMDevice::Update` حتى يقوم بنقل تلك الصورة الجديدة الجاهز للعرض إلى الشاشة.
- 5.

```

////////////////////////////////////
////////////////////////////////////
//
// RenderLoop
// Clear the viewport, render the next frame, and update the
// window.
//
////////////////////////////////////
////////////////////////////////////

static BOOL
RenderLoop()
{
    HRESULT rval;

    // Tick the scene.

    rval = myglobs.scene->lpVtbl->Move(myglobs.scene,
    D3DVAL(1.0));
    if (rval != D3DRM_OK) {
        return FALSE;
    }

    // Clear the viewport.

    rval = myglobs.view->lpVtbl->Clear(myglobs.view);
    if (rval != D3DRM_OK) {

```

```

        return FALSE;
    }

    // Render the scene to the viewport.

    rval = myglobs.view->lpVtbl->Render(myglobs.view,
        myglobs.scene);
    if (rval != D3DRM_OK) {
        return FALSE;
    }

    // Update the window.

    rval = myglobs.dev->lpVtbl->Update(myglobs.dev);
    if (rval != D3DRM_OK) {
        return FALSE;
    }
    return TRUE;
}

```

إنشاء المشهد : Creating the Scene

بعد إعداد المحيط ثلاثي الأبعاد (3D-environment) واختيار سواقة الأداة المناسبة (device driver) ثم إنشاء الأداة ثلاثية الإبعاد (3D device) ووجهة النظر ووضع مستوى الشكل الخارجي للبرنامج (Render State) ... الخ يقوم برنامجنا بعدد من الوظائف (functions) لتأهيل هذا المحيط ثلاثي الأبعاد للتعامل مع الجسم ثلاثي الأبعاد المستخدم و كذلك للتعامل مع الإطارات والإضاءة وهي كما يلي:

- وظيفة MyScene
- وظيفة MakeMyFrames
- وظيفة MakeMyLights
- وظيفة SetMyPositions
- وظيفة MakeMyMesh
- وظيفة MakeMyWrap

• وظيفة AddMyTexture

ونقوم بشرحها على حسب ترتيب الظهور:

وظيفة MyScene :

وظيفة MyScene في مثالنا تماثل وظيفة BuildScene الموجودة في جميع أمثلة Direct3D الموجودة من ضمن المجموعة البرمجية لـ DirectX (DirectX SDK). ففي هذه الوظيفة نقوم بإظهار الأجسام مع أشكالها المميزة والإضاءة المطلوبة. وتقوم وظيفة MyScene باستدعاء عدة وظائف معرفة محليا (locally defined) تقوم على أساسها بإنشاء صفات المشهد المبني وهي كما يلي:

• وظيفة MakeMyFrames

• وظيفة MakeMyLights

• وظيفة SetMyPositions

• وظيفة MakeMyMesh

• وظيفة MakeMyWrap

• وظيفة AddMyTexture

بعد أن تنتهي هذه الوظائف من إتمام الكائن المرئي (visual object) تقوم وظيفة MyScene باستدعاء الأمر IDirect3dRMFrame::AddVisual لإضافة الكائن المرئي إلى إطار المشهد. وبعد إضافة الكائن المرئي (visual object) إلى المشهد فإن وظيفة MyScene لا تحتاج إلى الواجهات التي أنشأتها لذلك تستدعي الأمر Release عدة مرات لإنهاء تلك الواجهات.

```

////////////////////////////////////
//
// MyScene
// Calls the functions that create the frames, lights, mesh, and
// texture. Releases all interfaces on completion.

```

```
//
/////////////////////////////////////////////////////////////////

BOOL
MyScene(LPDIRECT3DRMDEVICE dev, LPDIRECT3DRMVIEWPORT view,
        LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME lpCamera)
{
    LPDIRECT3DRMFRAME lpLightframe1 = NULL;
    LPDIRECT3DRMFRAME lpWorld_frame = NULL;
    LPDIRECT3DRMLIGHT lpLight1      = NULL;
    LPDIRECT3DRMLIGHT lpLight2      = NULL;
    LPDIRECT3DRMTEXTURE lpTex       = NULL;
    LPDIRECT3DRMWRAP lpWrap         = NULL;
    LPDIRECT3DRMMESHBUILDER lpSphere3_builder = NULL;

    MakeMyFrames(lpScene, lpCamera, &lpLightframe1, &lpWorld_frame);
    MakeMyLights(lpScene, lpCamera, lpLightframe1, &lpLight1,
                &lpLight2);
    SetMyPositions(lpScene, lpCamera, lpLightframe1, lpWorld_frame);
    MakeMyMesh(&lpSphere3_builder);
    MakeMyWrap(lpSphere3_builder, &lpWrap);
    AddMyTexture(lpSphere3_builder, &lpTex);

    // If you need to create a material (for example, to create
    // a shiny surface), call CreateMaterial and SetMaterial here.

    // Now that the visual object has been created, add it
    // to the world frame.

    lpWorld_frame->lpVtbl->AddVisual(lpWorld_frame,
                                     (LPDIRECT3DRMVISUAL) lpSphere3_builder);

    lpLightframe1->lpVtbl->Release(lpLightframe1);
    lpWorld_frame->lpVtbl->Release(lpWorld_frame);
    lpSphere3_builder->lpVtbl->Release(lpSphere3_builder);
    lpLight1->lpVtbl->Release(lpLight1);
    lpLight2->lpVtbl->Release(lpLight2);
    lpTex->lpVtbl->Release(lpTex);
    lpWrap->lpVtbl->Release(lpWrap);

    return TRUE;
}
```


وظيفة MakeMyFrames :

تستدعي وظيفة MyScene ووظيفة MakeMyFrames لإنشاء الإطارات للإضاءة المتجهة وإنشاء الإطار الرئيسي للمشهد. وتقوم بذلك عن طريق استدعاء الأمر .IDirect3DRM::CreatFrame

```

////////////////////////////////////
//////////
//
// MakeMyFrames
// Create frames used in the scene.
//
////////////////////////////////////
//////////

void MakeMyFrames (LPDIRECT3DRMFRAME lpScene,
LPDIRECT3DRMFRAME lpCamera,
LPDIRECT3DRMFRAME * lplpLightFrame1,
LPDIRECT3DRMFRAME * lplpWorld_frame)
{
    lpD3DRM->lpVtbl->CreateFrame (lpD3DRM, lpScene,
    lplpLightFrame1);
    lpD3DRM->lpVtbl->CreateFrame (lpD3DRM, lpScene,
    lplpWorld_frame);
}

```

وظيفة MakeMyLights :

تستدعي وظيفة MyScene ووظيفة MakeMyLights لإنشاء الإضاءة الطيفية والمتجهة في المشهد . تستدعي وظيفة MakeMyLights الأمر IDirect3DRMFrame:AddLight و IDirect3DRM::CreateLightRGB للإضاءة المتجهة وإضافتها إلى إطار الإضاءة وكذلك لإنشاء الإضاءة الطيفية وإضافتها للإطار الرئيسي (الإضاءة الطيفية تنسب إلى الإطار الرئيسي للمشهد في كل الأحوال).

```

////////////////////////////////////
////////
//
// MakeMyLights
// Create lights used in the scene.
//
////////////////////////////////////
////////

void MakeMyLights(LPDIRECT3DRMFRAME lpScene,
LPDIRECT3DRMFRAME lpCamera,
    LPDIRECT3DRMFRAME lpLightFrame1,
    LPDIRECT3DRMLIGHT * lpLight1, LPDIRECT3DRMLIGHT *
lpLight2)
{
    lpD3DRM->lpVtbl->CreateLightRGB(lpD3DRM,
D3DRMLIGHT_DIRECTIONAL,
    D3DVAL(0.9), D3DVAL(0.9), D3DVAL(0.9), lpLight1);

    lpLightFrame1->lpVtbl->AddLight(lpLightFrame1,
*lpLight1);

    lpD3DRM->lpVtbl->CreateLightRGB(lpD3DRM,
D3DRMLIGHT_AMBIENT,
    D3DVAL(0.1), D3DVAL(0.1), D3DVAL(0.1), lpLight2);

    lpScene->lpVtbl->AddLight(lpScene, *lpLight2);
}

```

وظيفة SetMyPositions :

تستدعي وظيفة MyScene ووظيفة SetMyPositions لوضع اتجاه و موقع الإطارات المستخدمة في المشهد. تقوم الوظيفة SetMyPositions بذلك عن طريق استدعاء الأمر IDirect3DRMFrame::SetPosition لوضع الموقــــــــــــــــع والأمرر IDirect3DRMFrame::SetOrientation لوضع الاتجاه وأخيرا استدعاء أمر درجة الدوران IDirect3DRMFrame::SetRotation كما يلي :

```

////////////////////////////////////
////////
//
// SetMyPositions
// Set the positions and orientations of the light, camera,
// and world frames. Establish a rotation for the globe.
//
////////////////////////////////////
////////

void SetMyPositions(LPDIRECT3DRMFRAME lpScene,
    LPDIRECT3DRMFRAME lpCamera, LPDIRECT3DRMFRAME
lpLightFrame1,
    LPDIRECT3DRMFRAME lpWorld_frame)
{

    lpLightFrame1->lpVtbl->SetPosition(lpLightFrame1,
lpScene, D3DVAL(2), D3DVAL(0.0), D3DVAL(22));

    lpCamera->lpVtbl->SetPosition(lpCamera, lpScene,
    D3DVAL(0.0), D3DVAL(0.0), D3DVAL(0.0));
    lpCamera->lpVtbl->SetOrientation(lpCamera, lpScene,
    D3DVAL(0.0), D3DVAL(0.0), D3DVAL(1),
    D3DVAL(0.0), D3DVAL(1), D3DVAL(0.0));

    lpWorld_frame->lpVtbl->SetPosition(lpWorld_frame,
lpScene, D3DVAL(0.0), D3DVAL(0.0), D3DVAL(15));
    lpWorld_frame->lpVtbl->SetOrientation(lpWorld_frame,
lpScene, D3DVAL(0.0), D3DVAL(0.0), D3DVAL(1),
    D3DVAL(0.0), D3DVAL(1), D3DVAL(0.0));

    lpWorld_frame->lpVtbl->SetRotation(lpWorld_frame,
lpScene, D3DVAL(0.0), D3DVAL(0.1), D3DVAL(0.0), D3DVAL(0.05));
}

```

وظيفة MakeMyMesh :

تستدعي وظيفة MyScene ووظيفة MakeMyMesh لتعبئة وإعداد الجسم ثلاثي الإبعاد (الكرة في هذه الحالة) ، فعن طريق الأمر IDirect3DRM::CreateMeshBuilder يقوم بإنشاء واجهة بنية الأجسام التي عن

طريقها نستخدم الأوامر Load() و Scale() و SetColorRGB() لإعداد الجسم ثلاثي الأبعاد والموجود على الملف Sphere3.X.

```

////////////////////////////////////
////////
//
// MakeMyMesh
// Create MeshBuilder object, load, scale, and color the
mesh.
//
////////////////////////////////////
////////

void MakeMyMesh(LPDIRECT3DRMMESHBUILDER *
lpSphere3_builder)
{
    lpD3DRM->lpVtbl->CreateMeshBuilder(lpD3DRM,
lpSphere3_builder);

    (*lpSphere3_builder)->lpVtbl-
>Load(*lpSphere3_builder,
    "sphere3.x", NULL, D3DRMLOAD_FROMFILE, NULL, NULL);

    (*lpSphere3_builder)->lpVtbl-
>Scale(*lpSphere3_builder,
    D3DVAL(2), D3DVAL(2), D3DVAL(2));

    // Set sphere to white to avoid unexpected texture-
blending
    // results.

    (*lpSphere3_builder)->lpVtbl-
>SetColorRGB(*lpSphere3_builder,
    D3DVAL(1), D3DVAL(1), D3DVAL(1));
}

```

وظيفة MakeMyWrap :

تستدعي وظيفة MyScene ووظيفة MakeMyWrap لإنشاء واستخدام الشكل الخارجي (texture) مع الجسم الثلاثي الأبعاد (الكرة) حيث تقوم باستدعاء أحد

أوامر واجهة بنية الأجسام IDirect3DRMMeshBuilder::GetBox للحصول على أبعاد الصندوق المحيط بالجسم (بالكرة في هذه الحالة) ويستخدم هذا الأمر للحصول على أبعاد أصغر صندوق يمكن احتواء ذلك الجسم داخله ونستخدم تلك الأبعاد لوضع الشكل الخارجي) وتستخدم تلك الأبعاد مع الأمر IDirect3DRM::CreateWrap لإنشاء الغلاف الخارجي للشكل الخارجي للجسم بعد ذلك تستدعي أحد أوامر واجهة الغلاف الخارجي للجسم IDirect3DRMWrap::Apply لوضع الشكل الخارجي على الجسم.

```

////////////////////////////////////
//////////
//
// MakeMyWrap
// Creates and applies wrap for texture.
//
////////////////////////////////////
//////////

void MakeMyWrap(LPDIRECT3DRMMESHBUILDER sphere3_builder,
               LPDIRECT3DRMWRAP * lpWrap)
{
    D3DVALUE miny, maxy, height;
    D3DRMBOX box;

    sphere3_builder->lpVtbl->GetBox(sphere3_builder, &box);

    maxy = box.max.y;
    miny = box.min.y;
    height = maxy - miny;

    lpD3DRM->lpVtbl->CreateWrap
        (lpD3DRM, D3DRMWRAP_CYLINDER, NULL,
         D3DVAL(0.0), D3DVAL(0.0), D3DVAL(0.0),
         D3DVAL(0.0), D3DVAL(1.0), D3DVAL(0.0),
         D3DVAL(0.0), D3DVAL(0.0), D3DVAL(1.0),
         D3DVAL(0.0), D3DDivide(miny, height),
         D3DVAL(1.0), D3DDivide(-D3DVAL(1.0), height),
         lpWrap);

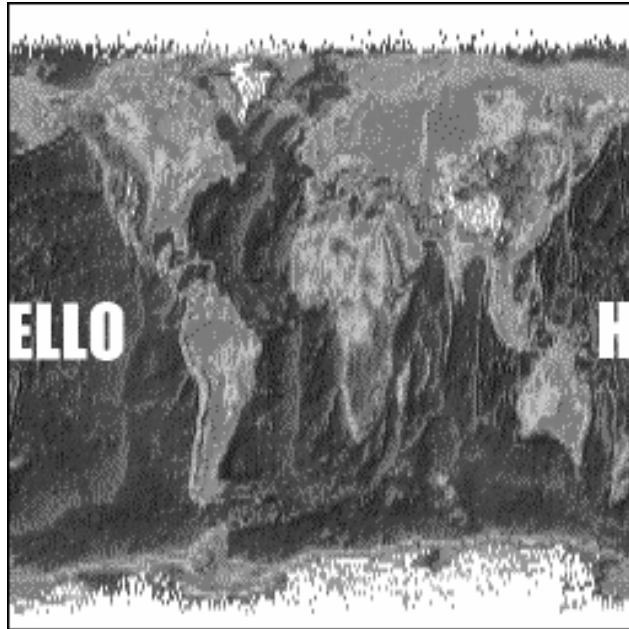
    (*lpWrap)->lpVtbl->Apply(*lpWrap, (LPDIRECT3DRMOBJECT)
        sphere3_builder);
}

```

}

وظيفة AddMyTexture :

تستدعي وظيفة MyScene وظيفة AddMyTexture لتعبئة الشكل الخارجي والحاقة بالجسم (الكرة). وتقوم هذه الوظيفة باستدعاء الأمر IDirect3DRM::LoadTexture لتعبئة صورة الشكل الخارجي (هذه الصورة هي tutor.bmp) بعد ذلك تستدعي الأمر IDirect3DRMMeshBuilder::SetTexture لإلحاق تلك الصورة بالجسم.



```
////////////////////////////////////  
//  
// AddMyTexture  
// Creates and applies wrap for texture.  
//  
////////////////////////////////////  
  
void AddMyTexture(LPDIRECT3DRMMESHBUILDER lpSphere3_builder,  
                 LPDIRECT3DRMTEXTURE * lpIpTex)  
{  
    lpD3DRM->lpVtbl->LoadTexture(lpD3DRM, "tutor.bmp", lpIpTex);  
  
    // If you need a color depth other than the default (16),  
    // call IDirect3DRMTexture::SetShades here.  
  
    lpSphere3_builder->lpVtbl->SetTexture(lpSphere3_builder,  
    *lpIpTex);  
}
```

إنهاء البرنامج Cleaning Up :

وأخيراً وظيفة إنهاء البرنامج تقوم بالتنفيذ عندما يستقبل البرنامج رسالة التدمير WM_DESTROY من ويندوز أو بعد عدة محاولات فاشلة لاستدعاء وظيفة

.RenderLoop

```
////////////////////////////////////  
//  
// CleanUp  
// Release all D3DRM objects and set the bQuit flag.  
//  
////////////////////////////////////  
  
void  
CleanUp(void)  
{  
    myglobs.bInitialized = FALSE;  
    myglobs.scene->lpVtbl->Release(myglobs.scene);  
    myglobs.camera->lpVtbl->Release(myglobs.camera);  
    myglobs.view->lpVtbl->Release(myglobs.view);  
    myglobs.dev->lpVtbl->Release(myglobs.dev);  
    lpD3DRM->lpVtbl->Release(lpD3DRM);  
    lpDDClipper->lpVtbl->Release(lpDDClipper);  
  
    myglobs.bQuit = TRUE;  
}
```


الفصل السادس عشر

مرجع أواخر المكتبة

AGDX

مرجع أوامر المكتبة AGDX

المكتبة AGDX عبارة عن كائنات (Object Classes) نقوم من خلالها بالتحكم ببرمجة تكنولوجيا DirectX ، وقد بُنيت كطبقة إضافية على كائنات DirectX نفسها. لكنها تتميز بإضافة سهولة الاستخدام لبرمجة هذه التكنولوجيا وتختلف عن كائنات هذه التكنولوجيا بأنها توفر لنا كل الميزات المطلوبة لإنشاء كل برامج الصوت والصورة بشكل عام والألعاب ذات الخلفيات المتحركة ، وتحكم نقطي بالشاشة ، واستخدام صور التايلز ، واستخدام الخلفيات الكبيرة الحجم واستخدام الممثلين ... الخ بشكل خاص. الهدف العام من هذه المكتبة هو جعل برمجة DirectX خفي عن المبرمج. في هذا الفصل سنقوم بشرح كل أعضاء (الأوامر) كائنات هذه المكتبة.

لأهمية اللغة الإنجليزية في البرمجة ستجد أحيانا في هذا المرجع شرح باللغتين ، العربية والإنجليزية. وكذلك للملاحظة أنك لو حاولت فهم هذا المرجع مباشرة دون قراءة الأمثلة التابعة لهذه المكتبة في هذا الكتاب فإنك لن تتعرف على الكثير من المصطلحات الموجودة ، وذلك لأنها وجدت هنا كمرجع فقط. وهناك شرح في الأمثلة لطريقة عمل هذه الأوامر. لذلك استخدم هذا الفصل كمرجع وليس كأساس لفهم مكتبة AGDX

AGDXScreen

AGDXScreen هو كائن الشاشة وهو الكائن الرئيسي لهذه المكتبة وكل برنامج يستخدم **AGDX** يجب أن يشتمل على كائن **AGDXScreen**. وكما نرى من أسم هذا الكائن فإنه يحتوي على أوامر ضرورية للتحكم في ذاكرة العرض وكذلك أوامر للتحكم في الألوان.

Data members

The DirectDraw object	m_lpDD
A AGDXSurface object for the front buffer	m_lpDDSFfront
A AGDXSurface object for the back buffer	m_lpDDSBback
The DirectDrawPalette object	m_lpDDPalette
The screen width in pixels	m_dwPixelWidth
The screen height in pixels	m_dwPixelHeight
The bits per pixel (number of colours)	m_BPP
Is the program active? Not currently used	m_bActive
Is the program running in full screen mode?	m_bFullScreen
The DirectDraw clipper, used in windowed mode only.	m_lpClipper

المتغيرات (أو المؤشرات) أعضاء هذا العنصر :

كائن DirectDraw	m_lpDD
كائن AGDXSurface للسطح الرئيسي	m_lpDDSFfront
كائن AGDXSurface للسطح الخلفي	m_lpDDSBback
كائن DirectDrawPalette	m_lpDDPalette
عرض الشاشة بعدد النقاط	m_dwPixelWidth
ارتفاع الشاشة بعدد النقاط	m_dwPixelHeight
البت لكل نقطة (عدد الألوان)	m_BPP
إذا كان البرنامج في حالة تشغيل (ليس مستعمل)	m_bActive
هل البرنامج يعمل على الشاشة بكاملها ؟	m_bFullScreen
كائن DirectDraw clipper يستعمل فقط لو أن البرنامج يعمل كنافذة.	m_lpClipper

Construction / Destruction

البناني و المدمر

AGDXScreen(void)

AGDXScreen(void *hWnd, DWORD Width, DWORD Height, DWORD BPP)

هذا البناني يقوم بإنشاء كائن DirectDraw ، ووضع الشاشة بالبعد النقطي المطلوب ولصق الأسطح (الأمامي والخلفي) ببعضها البعض ولا يختلف هذا البناني عن الأمر

CreateFullScreen()

AGDXScreen(void *hWnd, int Width, int Height)

هذا البناني يقوم بإنشاء كائن DirectDraw ، وتشغيل البرنامج كنافذة من نوافذ نظام التشغيل ولا يختلف هذا البناني عن الأمر

CreateWindowed()

~AGDXScreen()

Destroys the DirectDraw object and returns control to Windows.

هذا المدمر يقوم بتدمير كائن DirectDraw وإرجاع التحكم لنظام التشغيل.

Member functions

الأوامر الأعضاء

void CreateFullScreen(void *hWnd, DWORD Width, DWORD Height, DWORD BPP);

Creates the DirectDraw object, sets the screen resolution and creates and attaches the front and back buffer.

يقوم هذا الأمر بإنشاء كائن DirectDraw ووضع الشاشة بالبعد النقطي المطلوب ولصق الأسطح (الأمامي والخلفي) ببعضها البعض.

void CreateWindowed(void *hWnd, int Width, int Height);

Creates the DirectDraw object, initialises AGDX for a windowed mode application. The Width and Height are used to create the back buffer size, not the window size.

هذا الأمر يقوم بإنشاء كائن DirectDraw ، ووضع AGDX لتعمل كنافذة من نوافذ نظام التشغيل. العرض والارتفاع يستعملان لإنشاء السطح الخلفي وليس حجم النافذة.

void LoadBitmap(LPCSTR szFilename)

Loads a .BMP file straight onto the back buffer.

يقوم هذا الأمر بتعبئة ملف صورة من نوع BMP مباشرة إلى السطح الخلفي.

void LoadPalette(LPCSTR szFilename)

Loads the palette from a .BMP file. Should be called before other bitmap operations.

يقوم هذا الأمر بتعبئة الألوان من صورة ملف من نوع BMP في حالة استخدام 256 لون للشاشة يجب أن نستخدم هذا الأمر قبل أي عملية نقوم بها لإظهار الصور على الشاشة.

void **Fill**(DWORD FillColor)

Fills the back buffer with the specified colour.

يقوم هذا الأمر بتعبئة السطح الخلفي بلون معين.

HRESULT **Flip**(void)

Displays the contents of the back buffer on screen. If in a windowed mode, Flip will stretch the back buffer to fit into the window.

يقوم هذا الأمر بقلب محتويات السطح الخلفي إلى السطح الرئيسي والظاهر على الشاشة. في حالة لو أن البرنامج يعمل كنافذة يقوم هذا الأمر بتعديل حجم السطح الخلفي ليناسب حجم النافذة.

void **Restore**(void)

Restores the DirectDraw object if lost. Called internally if a flip fails.

في حاله فقدان كائن DirectDraw يقوم هذا الأمر باسترجاعه ، تقوم مكتبة AGDX باستدعاء هذا الأمر بشكل أتوماتيكي إذا لم تنجح عملية القلب في الأمر أعلاه.

void **SetColor**(int col, int r, int g, int b)

Sets the red, green and blue values of a single colour in the current palette.

عن طريق هذا الأمر نستطيع أن نغير قيمة أي لون معين في لوحة الألوان عن طريق القيم (الأحمر و الأخضر و الأزرق).

void **GetColor**(int col, int *r, int *g, int *b)

Returns the red, green and blue values of a single colour in the current palette.

عن طريق هذا الأمر نستطيع الحصول على قيمة أي لون معين في لوحة الألوان عن طريق القيم (الأحمر و الأخضر و الأزرق).

void **SetPalette**(int Start, int Count, LPPALETTEENTRY lpPE)

Sets the palette pointed to by lpPE. Start indicates the first entry to be set and Count is the number of palette entries to be changed.

نستطيع عن طريق هذا الأمر أن نستعمل ملف ألوان مختلف عن الحالي والذي يؤشر إليه المتغير lpPE. المتغير Start يمثل أول لون يجب تغييره والمتغير Count يمثل عدد الألوان المطلوب تغييرها.

void **GetPalette**(int Start, int Count, LPPALETTEENTRY lpPE)

Fills lpPE with values from the current palette. Start indicates the first entry to be set and Count is the number of palette entries to be changed.

نستطيع عن طريق هذا الأمر أن نستعمل ملف الألوان الحالي لتغيير ألوان ملف مختلف والذي يؤشر إليه المتغير lpPE. المتغير Start يمثل أول لون يجب تغييره والمتغير Count يمثل عدد الألوان المطلوب تغييرها.

void **FillPalette**(int r, int g, int b)

Sets all the palette entries to a single colour.

يقوم هذا الأمر بتعبئة ملف الألوان بلون معين واحد يرمز إليه بالقيم r, g, b

void **GreyScale**(void)

Converts the current palette into a monochrome palette.

حول ملف الألوان الحالي إلى ملف أبيض واسود بحيث تظهر لنا الشاشة بدون ألوان.

void **FadeIn**(int delay, LPPALETTEENTRY lpPE)

يقوم هذا الأمر وبشكل سلس بتغيير لوحة الألوان الحالية إلى لوحة الألوان المشار إليها عن طريق المتغير lpPE. ولكي نقوم بذلك يجب أن نتبع ثلاث خطوات. الخطوة الأولى هي الحصول على ملف لوحة الألوان الحالية باستخدام الأمر GetPalette بعد ذلك وفي الخطوة الثانية يجب أن نضع ملف لوحة الألوان الحالي للون الذي نحب أن نبدأ منه وأخير في الخطوة الثالثة نستخدم هذا الأمر FadeIn مع ملف لوحة الألوان الذي حصلنا عليه في الخطوة الأولى.

لننظر إلى هذا المثال :

```
AGDXScreen Screen;
PALETTEENTRY pe[256];
...
```

```
Screen->GetPalette(0, 256, pe);
Screen->FillPalette(255, 0, 0);
Screen->FadeIn(6, pe);
```

هذا المثال سوف يجعل الشاشة تصبح باهته باللون الأحمر ثم ترجع كما كانت ، مثل لعبة Quake عندما تُضرب بصاروخ.

void **FadeOut**(int delay)

Smoothly fades the current palette to black. A delay of 0 to 10 seems about right (0 - fast, 10 - slow).

يقوم هذا الأمر وبشكل سلس بتغيير الألوان الحالية على الشاشة إلى اللون الأسود. عملية التغيير هذا تعتمد على قيمة المتغير delay بحيث القيمة 0 تعطي تغيير سريع و القيمة 10 تعطي تغيير بطيء.

```
void FadeTo(int r, int g, int b, int delay)
```

يقوم هذا الأمر وبشكل سلس بتغيير الألوان الحالية على الشاشة إلى اللون المشار إليه عن طريق القيم **r** (اختصار كلمة red ويرمز للقيمة الحمراء للون) و **g** (اختصار كلمة green ويرمز للقيمة الخضراء للون) و **b** (اختصار كلمة blue ويرمز للقيمة الزرقاء للون). عملية التغيير هذا تعتمد على قيمة المتغير delay بحيث القيمة 0 تعطي تغيير سريع و القيمة 10 تعطي تغيير بطيء.

```
LPDIRECTDRAW GetDD(void) { return m_lpDD; }
AGDXSurface* GetFront(void) { return m_lpDDFront.m_lpDDS; }
AGDXSurface* GetBack(void) { return m_lpDDSBack.m_lpDDS; }
LPDIRECTDRAWPALETTE GetPalette(void) { return m_lpDDPalette; }
```

هذه الأوامر تقوم بإرجاع المؤشرات لكائناتها. مثلاً الأمرين GetFront (أي احصل على السطح الرئيسي) و GetBack (أي احصل على السطح الخلفي) يقومان بإرجاع المؤشر إلى السطح المطلوب في كائن AGDXSurface.

AGDXSurface

هذا هو الكائن المسؤول عن الأسطح (وهو نسخة مطورة من كائن الأسطح الأصلي IDirectDrawSurface التابع لتكنولوجيا DirectX) ويقوم بالاحتفاظ بالمعلومات عن الصور التي سيتم عرضها على الشاشة. ونستخدم هذا الكائن كأساس لكثير من الكائنات في هذه المكتبة AGDX. ومن الأشياء التي يجب معرفتها أننا في تكنولوجيا DirectX لا نستطيع إنشاء أسطح عرضها اكبر من عرض الشاشة.

Data members

	m_PixelWidth	The surface width in pixels
	m_PixelHeight	The surface height in pixels
m_pFilename		The name of the bitmap file loaded, used in Restore
SrcRect		A RECT structure holding the surface source
DestRect		A RECT structure holding the surface destination
	m_lpDDS	The LPDIRECTDRAW_SURFACE object
	Screen	A pointer to a AGDXScreen object
m_DC		A HDC object, used for drawing text to the surface
m_DDSD		A DDSURFACEDESC object, stores surface information
m_Font		A HFONT object describing the currently selected font
	m_ColorKey	The specified color key for this surface

المتغيرات (أو المؤشرات) أعضاء هذا العنصر :

عرض السطح بعدد النقاط	m_PixelWidth
ارتفاع السطح بعدد النقاط	m_PixelHeight
اسم الصورة التي تم تعيينها ، يستخدم في حالة فقدان السطح	m_pFilename
متغير من نوع struct يحمل حجم السطح المرسل	SrcRect
متغير من نوع struct يحمل حجم السطح المرسل إليه	DestRect
كائن LPDIRECTDRAW_SURFACE	m_lpDDS
مؤشر إلى كائن AGDXScreen	Screen
كائن HDC يستخدم لرسم الكتابة على السطح	m_DC
كائن DDSURFACEDESC2 ويحتفظ بمعلومات عن السطح	m_DDSD
كائن HFONT يحتفظ بموصفات البنط المختار حالياً	m_Font
يحمل هذا المتغير قيمة اللون الشفاف (المفتاح) لهذا السطح.	m_ColorKey

Construction / Destruction الباني و المدمر**AGDXSurface()****AGDXSurface**(AGDXScreen *pScreen, char *szFilename)

Creates a surface holding the bitmap pointed to by szFilename.

يقوم هذا الباني بإنشاء سطح يحتفظ بصورة معينة مشار لها بالمؤشر szFilename

AGDXSurface(AGDXScreen *pScreen, int Width, int Height)

Creates a empty surface with a size set by Width and Height.

يقوم هذا الباني بإنشاء سطح بحجم ذو عرض معين محدد بقيمة المتغير Width وارتفاع معين محدد بقيمة المتغير Height.

~AGDXSurface()

Destroys the surface and frees the memory.

يقوم هذا المدمر بتدمير السطح وتحرير الذاكرة المحجوزة له.

Member functions

الأوامر الأعضاء

void **Create**(AGDXScreen *pScreen, char *szFilename)

Creates a surface holding the bitmap pointed to by szFilename.

يقوم هذا الأمر بإنشاء سطح يحتفظ بصورة معينة مشار لها بالمؤشر szFilename

void **Create**(AGDXScreen *pScreen, int Width, int Height)

Creates a empty surface with a size set by Width and Height.

يقوم هذا الأمر بإنشاء سطح بحجم ذوا عرض معين محدد بقيمة المتغير Width وارتفاع معين محدد بقيمة المتغير Height.

void **SetDest**(int t, int l, int b, int r)

Sets the destination for a Draw operation.

يقوم هذا الأمر بوضع الهدف لعملية الرسم والتي نقوم بها عن طريق أوامر Draw

void **SetSrc**(int t, int l, int b, int r)

Sets the source for a Draw operation.

يقوم هذا الأمر بوضع المصدر لعملية الرسم والتي نقوم بها عن طريق أوامر Draw

void **ColorKey**(int col)

يقوم هذا الأمر بوضع اللون المفتاح والمشار إليه بالمتغير COI . هذا اللون سيكون اللون الشفاف عند عرضه على الشاشة.

void **Restore**(void)

في حالة فقدان كائن DirectDrawSurface والصورة الموجودة به يقوم هذا الأمر باسترجاعه ، تقوم مكتبة AGDX باستدعاء هذا الأمر بشكل أتوماتيكي إذا لم تنجح عملية ارسم عن طريق أوامر Draw.

virtual HRESULT **Draw**(AGDXSurface* lpDDS)

Copies the bitmap from the source surface to the surface pointed to by lpDDS. Uses the SrcRect and DestRect structures to position the bitmap. You can shrink and stretch surfaces with this method.

يقوم هذا الأمر بنسخ الصورة من السطح المصدر إلى السطح المشار إليه عن طريق المؤشر lpDDS ويستخدم الأمرين SrcRect و DestRect (سبق وشرحنا هذين الأمرين) لمعرفة الموقع الصحيح للصورة. تستطيع أن تكبر أو تصغر الأسطح بهذه الطريقة.

virtual HRESULT **DrawFast**(int X, int Y, AGDXSurface* lpDDS)

Copies the bitmap from the source surface to the surface pointed to by lpDDS. Uses X and Y to position the bitmap.

هذا أحد أوامر Draw (كل الأوامر التي تبدأ بهذه الكلمة Draw هي أوامر Draw) ويقوم بنسخ الصورة من السطح المصدر إلى السطح الهدف والمؤشر إليه عن طريق المؤشر lpDDS. ويستخدم المتغيرين X و Y لتحديد موقع الصورة.

```
virtual HRESULT DrawTrans(int X, int Y, AGDXSurface* lpDDS)
```

هذا أحد أوامر Draw ويقوم بنسخ الصورة من السطح المصدر إلى السطح الهدف والمؤشر إليه عن طريق المؤشر lpDDS. ويستخدم المتغيرين X و Y لتحديد موقع الصورة. وكذلك يستخدم اللون المفتاح كلون شفاف عندما يقوم بعملية النسخ.

```
virtual HRESULT DrawClipped(int X, int Y, AGDXSurface* lpDDS)
```

هذا أحد أوامر Draw ويقوم بنسخ الصورة من السطح المصدر إلى السطح الهدف والمؤشر إليه عن طريق المؤشر lpDDS. ويستخدم المتغيرين X و Y لتحديد موقع الصورة. وكذلك يستخدم اللون المفتاح كلون شفاف عندما يقوم بعملية النسخ. هذا الأمر يقوم بقص الصورة إذا خرجت أطرافها عن مساحة الشاشة وإلا فإن العنصر DirectDraw لن يكمل هذه العملية. إذا أردت أنت وعن قصد قص جزء من الصورة في داخل الشاشة نفسها استخدم الأمر Clip بنفسك (في حالة لو أنك تريد توزيع الشاشة لمناطق معينة مثلاً).

```
virtual HRESULT DrawWindowed(AGDXSurface* lpDDS)
virtual HRESULT DrawScaled(int X, int Y, float Factor, AGDXSurface* lpDDS)
virtual void DrawRotated(int X, int Y, double Angle, AGDXSurface* lpDDS)
virtual void DrawHFlip(int X, int Y, AGDXSurface* lpDDS)
virtual void DrawVFlip(int X, int Y, AGDXSurface* lpDDS)
```

كل هذه أوامر Draw وهي شبيهة للأوامر السابقة وما تقوم به مشار إليه عن طريق أسمائها.

```
void Lock(void)
```


عندما نستخدم هذا الأمر فإننا نحصل على تحكم مباشر في السطح والموجود في ذاكرة العرض. ويجب استخدام هذا الأمر قبل استخدام أحد أوامر الرسم مثل PutPixel وهي وضع نقطة في مكان معين على الشاشة أو Rect وهو رسم مستطيل على الشاشة أو Line وهو رسم خط على الشاشة... الخ. ولكن تذكر أنه عليك استخدام الأمر التالي UnLock عند انتهائك من استخدام أوامر الرسم.

void **UnLock**(void)

يقوم هذا الأمر بحماية السطح من أي تغيير ويجب أن نستخدم هذا الأمر مباشرة بعد أن ننتهي من استخدام أوامر الرسم والتي تتبع الأمر السابق Lock.

void **GetDC**(void)

Retrieves a handle of the display device context (DC) and stores it in the m_DC data member. The device context is used in GDI functions to draw to the screen. Also used by the **TextXY** function.

عن طريق هذا الأمر نستطيع الحصول على عنوان DC وهو ما نستخدمه windows للرسم بدون استخدام تكنولوجيا DirectX بعد الحصول على هذا العنوان يتم تخزينه في المتغير m_DC.

ونستخدم الـ DC عندما نريد استخدام أوامر الـ GDI للرسم على الشاشة. ونستخدمه كذلك عندما نستخدم أمر الكتابة على الشاشة TextXY.

void **ReleaseDC**(void)

Releases the device context. Should be called after you have finished with a **GetDC** function.

عن طريق هذا الأمر نقوم بتحرير الذاكرة التي حجزناها للـ DC ويحب استخدام هذا الأمر بعدما ننتهي من عملنا مع الأمر السابق GetDC.

void ChangeFont(char* FontName, int Width, int Height, int Attributes = FW_NORMAL)

نستخدم هذا الأمر عندما نريد اختيار بنط الكتابة مع الأمر TextXY. المتغير FontName يمثل اسم البنط المطلوب ويجب أن يكون من ضمن الأبناط المستخدمة من قبل نظام التشغيل (ملاحظة أنظر في ملف الأبناط لديك لتجد قائمة بأسماء الأبناط المستخدمة مثلا "Comic Sans MS") العرض Width والارتفاع Height تحدد عرض وارتفاع كل حرف بعدد النقاط. وأخيرا المتغير Attributes يشير إلى وزن البنط من 0 إلى 1000 ، مثلا الوزن عادي والوزن 700 يجعل البنط سميك.

void SetFont(void)

عن طريق هذا الأمر نقوم باستخدام البنط الذي سبق واخترناه في الأمر السابق ChangeFont. تذكر أن تستخدم الأمر GetDC قبل أن تستخدم هذا الأمر.

void TextXY(int X, int Y, COLORREF Col, LPCTSTR pString)

نستخدم هذا الأمر لكتابه نص معين (يكون موجود في المتغير pString) على السطح المطلوب في الموقع X و Y وبلون Col. تذكر أن تستخدم الأمر GetDC قبل أن تستخدم هذا الأمر.

void PutPixel(int X, int Y, int Col)

نستخدم هذا الأمر لرسم نقطة على السطح المطلوب في الموقع X و Y وبلون Col. هذا الأمر يُستخدم فقط عندما يكون مجموع الألوان على الشاشة 256 (بمعنى 8 بت) تذكر أن تستخدم الأمر Lock قبل أن تستخدم هذا الأمر.

int GetPixel(int X, int Y)

نستخدم هذا الأمر للحصول على لون نقطة على السطح المطلوب في الموقع X و Y. هذا الأمر يُستخدم فقط عندما يكون مجموع الألوان على الشاشة 256 (بمعنى 8 بت) تذكر أن تستخدم الأمر **Lock** قبل أن تستخدم هذا الأمر.

```
void Rect(int X1,int Y1,int X2,int Y2,int Col)
```

نستخدم هذا الأمر لرسم مستطيل على السطح المطلوب في الموقع أعلى اليسار X1 و Y1 وأسفل اليمين X2 و Y2 وبلون Col. هذا الأمر يُستخدم فقط عندما يكون مجموع الألوان على الشاشة 256 (بمعنى 8 بت) تذكر أن تستخدم الأمر **Lock** قبل أن تستخدم هذا الأمر.

```
void FillRect(int X1, int Y1, int X2, int Y2, int Col)
```

نستخدم هذا الأمر لرسم مستطيل معيّن بلون معين على السطح المطلوب في الموقع أعلى اليسار X1 و Y1 وأسفل اليمين X2 و Y2 وبلون Col. هذا الأمر يُستخدم فقط عندما يكون مجموع الألوان على الشاشة 256 (بمعنى 8 بت) تذكر أن تستخدم الأمر **Lock** قبل أن تستخدم هذا الأمر.

```
void Line(int X1, int Y1, int X2, int Y2, int Col)
```

نستخدم هذا الأمر لرسم خط مستقيم على السطح المطلوب يبدأ من الموقع X1 و Y1 وينتهي عند الموقع X2 و Y2 وبلون Col. هذا الأمر يُستخدم فقط عندما يكون مجموع الألوان على الشاشة 256 (بمعنى 8 بت) تذكر أن تستخدم الأمر **Lock** قبل أن تستخدم هذا الأمر.

```
void VLine(int Y1, int Y2, int X, int Col)
```

نستخدم هذا الأمر لرسم خط عمودي مستقيم على السطح المطلوب يبدأ من الموقع X و Y1 وينتهي عند الموقع X و Y2 وبلون Col. هذا الأمر يُستخدم فقط عندما يكون مجموع الألوان على الشاشة 256 (بمعنى 8 بت) تذكر أن تستخدم الأمر **Lock** قبل أن تستخدم هذا الأمر.

void **HLine**(int X1, int X2, int Y, int Col)

نستخدم هذا الأمر لرسم خط أفقي مستقيم على السطح المطلوب يبدأ من الموقع X1 و Y وينتهي عند الموقع X2 و Y وبلون Col. هذا الأمر يُستخدم فقط عندما يكون مجموع الألوان على الشاشة 256 (بمعنى 8 بت) تذكر أن تستخدم الأمر **Lock** قبل أن تستخدم هذا الأمر.

void **Circle**(int X, int Y, int Radius, int Col)

نستخدم هذا الأمر لرسم دائرة على السطح المطلوب في الموقع X و Y ونصف قطر Radius وبلون Col. هذا الأمر يُستخدم فقط عندما يكون مجموع الألوان على الشاشة 256 (بمعنى 8 بت) تذكر أن تستخدم الأمر **Lock** قبل أن تستخدم هذا الأمر.

AGDXLayer

الكائن الطبقي AGDXLayer هو كائن وارث لكائن الأسطح AGDXSurface والذي تعرفنا عليه سابقا في هذا المرجع. وبالتالي فإنه يرث كل صفاته وأعضائه ولكن هذا الكائن AGDXLayer قد أضاف القدرة على تحريك الأسطح بشكل سلس في كل الاتجاهات. وتستطيع أن ترى هذه النوعية من المؤثرات في الألعاب التي تستخدم الخلفيات المتحركة بحيث تكون الخلفية عبارة عن صورة واحدة تقوم بتحريكها ونضع فوقها خلفيات التايلز المتحركة مما يعطي إحساس بالعمق. والكائن AGDXLayer هو اختيار ممتاز للألعاب التي تعتمد على الصور الطويلة عموديا (لعبة كرة الطاولة مثلا) لأن DirectX لا تحد من ارتفاع الأسطح في ذاكرة الفيديو.

Data members

The current X position in pixels	m_XOffset
The current Y position in pixels	m_YOffset

المتغيرات (أو المؤشرات) أعضاء هذا العنصر :

الموقع الحالي X ويحسب بعدد النقاط.	m_Xoffset
الموقع الحالي Y ويحسب بعدد النقاط.	m_Yoffset

Construction / Destruction الباني و المدمر

AGDXLayer(AGDXScreen *pScreen, char *szFilename)

Creates a DirectDrawSurface object holding the bitmap pointed to by szFilename.

هذا الباني يقوم بإنشاء سطح يحتفظ بصورة مؤشر لها عن طريق المؤشر szFilename.

~AGDXLayer()

Destroys the surface and frees the memory.

هذا المدمر يقوم بتدمير كائن **AGDXLayer**() وتحرير الذاكرة.

Member functions

الأوامر الأعضاء

void **ScrollUp**(int Offset)
 void **ScrollDown**(int Offset)
 void **ScrollLeft**(int Offset)
 void **ScrollRight**(int Offset)

These functions scroll the bitmap by the number of pixels indicated by Offset.

جميع هذه الأوامر تُستخدم لتحريك السطح وبالتالي الصورة في الاتجاه المطلوب وتحسب الحركة بعدد النقاط.

void **MoveTo**(int XOffset, int YOffset)
 Sets the position in the bitmap.

نستطيع عن طريق هذا الأمر وضع الموقع المطلوب في الصورة.

`virtual void Draw(LPDIRECTDRAWSURFACE lpDDS)`

Copies the bitmap from the source surface to the surface pointed to by lpDDS. Uses the DestRect structure to position the bitmap. The bitmap automatically wraps around the screen.

عن طريق هذا الأمر نقوم بنسخ صورة من سطح معين (المصدر) إلى سطح آخر مشار إليه بالمؤشر lpDDS. ونستخدم الأمر DestRect لنضع الموقع المطلوب للصورة. والصورة تقوم بتدوير نفسها بشكل تلقائي حول الشاشة.

`virtual void Draw(int X,int Y,AGDXSurface* lpDDS)`

Copies the bitmap from the source surface to the surface pointed to by lpDDS. And puts the upper corner of the bitmap in the x,y position Uses the DestRect structure to position the bitmap. The bitmap automatically wraps around the screen.

عن طريق هذا الأمر نقوم بنسخ صورة من سطح معين (المصدر) إلى سطح آخر مشار إليه بالمؤشر lpDDS. ونضع زاوية أعلى اليمين للصورة في الموقع x و y.

`virtual void DrawTrans(int X,int Y,AGDXSurface* lpDDS)`

Copies the bitmap from the source surface to the surface pointed to by lpDDS. And puts the upper corner of the bitmap in the x,y position Uses the DestRect structure to position the bitmap. The bitmap Automatically wraps around the screen. The only difference between this function and the last one is the transparent color for the layer.

عن طريق هذا الأمر نقوم بنسخ صورة من سطح معين (المصدر) إلى سطح آخر مشار إليه بالمؤشر lpDDS. ونضع زاوية أعلى اليمين للصورة في الموقع x و y. الفرق الوحيد بين هذه الوظيفة و الوظيفة السابقة هو أنها تستخدم مفتاح الألوان - يجب طبعا أن نكون قد حددنا مفتاح الألوان عند إنشاء هذه الكائن.

AGDXTile

هذا الكائن هو المسؤول عن خلفيات التايلز (وهي صور صغيرة مربعة الشكل شبيهة بالصور التي تستخدم لخلفيات صفحات الإنترنت) ويقوم هذا الكائن بالاحتفاظ بكل صور التايلز لخريطة الخلفية وكذلك بصور الممثلين المختلفة والتي تمثل في مجموعها الحركة. وهذا الكائن وارث للكائن AGDXSurface.

بعض الملاحظات المهمة : الصور التي تحتوي على مربعات التايلز يجب أن تحتوي على مربع تايل خالي، بمعنى أن أول صورة تايل يجب أن يكون لونها اسود (على أساس أن اللون الأسود هو اللون الشفاف) وذلك لأن AGDX تتعامل مع أول صورة تايل على أساس أنها تحتوي على اللون الشفاف. صور التايلز يجب أن تحفظ من اليسار إلى اليمين ومن الأعلى إلى الأسفل ويجب أن يكون مجموع عرضها يساوي عرض الصورة الحاملة لها. انظر إلى الأمثلة الكثيرة في هذا الكتاب إذا أحببت توضيح أكثر. وتذكر استخدام أمر اللون المفتاح colour key للسطح الذي يحمل صور التايلز.

Data members

The width of one tile, in pixels	m_BlockWidth
The height of one tile, in pixels	m_BlockHeight
The number of tiles in the bitmap file	m_BlockNum

المتغيرات (أو المؤشرات) أعضاء هذا العنصر :

m_BlockWid	عرض صورة تايل واحدة بعدد النقاط
m_BlockHeight	ارتفاع صورة تايل واحدة بعدد النقاط
m_BlockNum	عدد صور التايلز الموجودة في الصورة

Construction / Destruction الباني و المدمر
--

AGDXTile()

AGDXTile(AGDXScreen *pScreen, char* szFilename, int w, int h, int num)

Creates a DirectDrawSurface and loads the tiles from a .BMP file onto it. You must also specify the width and height of a single tile and the number of tiles in the bitmap.

يقوم هذا الباني بإنشاء سطح خلفي وتعبئته بصور التايلز الموجودة في صورة من نوع BMP. ويجب أن تحدد العرض والارتفاع لكل صورة تايل ومجموعها في الصورة.

~AGDXTile()

Destroys the surface and frees the memory.

يقوم هذا المدمر بتدمير السطح وتحرير الذاكرة.

Member functions

الأوامر الأعضاء

BOOL Create(AGDXScreen *pScreen, char* szFilename, int w, int h, int num)

هذا الأمر الوحيد لهذا الكائن ويقوم بإنشاء سطح خلفي وتعبئته بصور التايلز الموجودة في صورة من نوع BMP. ويجب أن تحدد العرض والارتفاع لكل صورة تايل ومجموعها في الصورة.

AGDXMap

This is the main object which controls tile based scrolling in AGDX. Screen, tile and map information all come together in this class to produce a scrolling output. Map information is stored in binary files in the simplest possible form, like so:

Map width (4 bytes)
Map height (4 bytes)
Map data (Map width * Map height * 4 bytes)

هذا الكائن الرئيسي في التحكم بحركة الخلفيات في هذه المكتبة. كائن الشاشة و كائن صور التايلز و كائن الخريطة المرافقة كلها تجتمع معاً هنا لتعطينا الخلفية المتحركة. المعلومات المحفوظة على خريطة الخلفية تكون من نوعية (binary أو ثنائي) بأبسط شكل ممكن ، انظر لما يلي :

عرض الخريطة (4 بايت)
ارتفاع الخريطة (4 بايت)
معلومات الخريطة (عرض الخريطة * ارتفاع الخريطة * 4 بايت)

انتبه: هذه ترجمة حرفية لأوامر الكائن الظاهرة في الأعلى وضعت فقط للشرح ويجب كتابتها كما هي باللغة الإنجليزية.

Data members

The current map X position in pixels	m_PosX
The current map Y position in pixels	m_PosY
The map width	m_Width
The map height	m_Height
The width of the map in pixels	m_PixelWidth
The height of the map in pixels	m_PixelHeight
Tile width in pixels, from AGDXTile pointer	m_TileWidth
Tile height in pixels, from AGDXTile pointer	m_TileHeight
Number of tiles wide on screen	SCREEN_TW
Number of tiles high on screen	SCREEN_TH

Screen pixel width, from AGDXScreen	SCREEN_W pointer
Screen pixel height, from AGDXScreen	SCREEN_H pointer
Map size, width * height	SIZE
The map data array	DATA
Pointer to a AGDXScreen object	Screen
Pointer to a AGDXTile object	Tiles

المتغيرات (أو المؤشرات) أعضاء هذا العنصر :

الموقع الحالي X للخريطة بعدد النقاط	m_PosX
الموقع الحالي Y للخريطة بعدد النقاط	m_PosY
عرض الخريطة	m_Width
ارتفاع الخريطة	m_Height
عرض الخريطة بعدد النقاط	m_PixelWidth
ارتفاع الخريطة بعدد النقاط	m_PixelHeight
عرض صورة التايل بعدد النقاط	m_TileWidth
ارتفاع صورة التايل بعدد النقاط	m_TileHeight
عدد صور التايل بشكل أفقي على الشاشة	SCREEN_TW
عدد صور التايل بشكل عمودي على الشاشة	SCREEN_TH
عرض الشاشة بالنقاط ونحصل عليه من المؤشر لكائن الشاشة	SCREEN_W
ارتفاع الشاشة بالنقاط ونحصل عليه من المؤشر لكائن الشاشة	SCREEN_H
حجم الخريطة = الارتفاع x العرض	SIZE
المتغير الحامل لمعلومات الخريطة	DATA
مؤشر لكائن الشاشة	Screen
مؤشر لكائن صور التايلز	Tiles

Construction / Destruction الباني و المدمر

AGDXMap(AGDXTile *pTiles, AGDXScreen *pScreen)
Constructs a map object. You must pass a pointer to the tiles you want the map to use and a pointer to a AGDXScreen object.

هذا هو الباني لكائن الخريطة وعن طريقة نمرر مؤشر إلى كائن التايلز AGDXTile وآخر لكائن الشاشة AGDXScreen.

~AGDXMap()
Destroys the map and frees the memory.

هذا المدمر وكما هي العادة نقوم في هذه الوظيفة بتدمير كائن الخريطة.

Member functions

الأوامر الأعضاء

void Create(int Width, int Height, int Fill)
Creates a new map.

أمر الإنشاء وعن طريقة نقوم بإنشاء خريطة بارتفاع معين وعرض معين ومفتاح لون (لون شفاف) معين.

BOOL Load(const char *szFilename)
Loads a map from a file.

يقوم هذا الأمر بتعبئة الملف الحامل للخريطة.

BOOL Save(const char *szFilename)
Saves a map to a file.

يقوم هذا الأمر بحفظ ملف الخريطة (يستخدم فقط في إنتاج محرر الخرائط)

void **Clear**(void)
Clears the map.

يقوم هذا الأمر بمسح جميع المعلومات الموجودة في الخريطة الحالية.

void **Fill**(int TileNum)
Fills the map with a tile specified by TileNum.
يقوم هذا الأمر بتعبئة الخريطة بصورة تايل ذات الرقم TileNum

void **Draw**(AGDXSurface* lpDDS)
Draws the map to the surface pointed to by lpDDS.
يقوم هذا الأمر برسم (وضع) الخريطة في السطح المشار إليه بالمؤشر lpDDS

void **DrawTrans**(AGDXSurface* lpDDS)
Draws the map to the surface pointed to by lpDDS with transparent tiles. If you haven't set the colour key for the tiles nothing will be drawn.

يقوم هذا الأمر برسم (وضع) الخريطة في السطح المشار إليه بالمؤشر lpDDS ولكن مع احترام اللون المفتاح (الشفاف) يجب طبعا أن نكون حددنا هذا اللون عند إنشاء الكائن.

void **DrawClipped**(AGDXSurface* lpDDS, LPRECT ClipDestRect)
Draws the map to the surface pointed to by lpDDS. The map is clipped to the rectangle ClipDestRect.
يقوم هذا الأمر برسم (وضع) الخريطة في السطح المشار إليه بالمؤشر lpDDS ولكنه أيضا يقوم بوضع الخريطة في المستطيل ClipDestRect

void **MoveTo**(int PosX, int PosY)
void **ScrollUp**(int Offset)
void **ScrollDown**(int Offset)
void **ScrollLeft**(int Offset)
void **ScrollRight**(int Offset)

This group of functions are used to set the position in the map. All values are in pixels.

هذه مجموعة الأوامر المسؤولة عن وضع موقع الخريطة وتحريكها وكلها تتعامل بنقاط الشاشة.

```
void WrapScrollUp(int Offset)
void WrapScrollDown(int Offset)
void WrapScrollLeft(int Offset)
void WrapScrollRight(int Offset)
```

This group of functions are used to scroll around the map. The map will wrap if you move outside the area of the map, e.g. If you scroll off the top of the map you will appear at the bottom.

هذه مجموعة الأوامر المسؤولة عن وضع موقع الخريطة وتحريكها وكلها تتعامل بنقاط الشاشة. وتختلف عن المجموعة السابقة في أن الخريطة ستقوم بتدوير نفسها بشكل التفاضلي ، مثلا لو حركنا الخريطة من الأعلى إلى الأسفل ، فعندما تختفي نقطة من الأسفل فإنها سوف تظهر في الأعلى و هلم جرا بشكل التفاضلي وبذلك تبدو وكأنها لا نهائية.

```
void ScreenTileSize(int Width, int Height)
Sets the screen tile width and screen tile height. This is the number of tiles drawn vertically and horizontally on screen.
```

يقوم هذا الأمر بوضع عدد صور التايل الأفقية و الرأسية الظاهرة على الشاشة. هذا عدد صور التايل التي ستُرسَم بشكل أفقي ورأسي على الشاشة.

```
void SetTile(int MapX, int MapY, int Tile)
Sets the tile at MapX, MapY to the value of Tile. Could be used for tile animation.
```

يقوم هذا الأمر بوضع صورة تايل معينه (نعبر عنها بقيمة المتغير Tile) في الموقع MapX و MapY. ونستطيع استخدام هذا الأمر لإنشاء الحركة لصور التايل.

```
int GetTile(int MapX, int MapY)
```

نستخدم هذا الأمر للحصول على قيمة صورة تايل معينه موجودة في الموقع MapX و MapY. ونستطيع استخدام هذا الأمر لاختبار التصادم.

```
void LoadTiles(AGDXTile *pTiles)
Loads a different set of tiles to the map.
يقوم هذا الأمر بتعبئة صور تايل جديدة إلى الخريطة
```

AGDXSprite

هذا الكائن يحتوي على الأوامر (أو الوظائف) المطلوبة للتحكم بالمتلين. ويقوم هذا الكائن بإنشاء مؤشر إلى الكائن AGDXTile والذي بدوره يقوم بتخزين الصور العديدة للمثل. وكل صور المثل يجب أن تكون بنفس الطول والعرض وموجودة في نفس الملف.

Data members

	m_PosX	The sprite's X position
	m_PosY	The sprite's Y position
	m_VelX	The sprite's X velocity
	m_VelY	The sprite's Y velocity
	m_Frame	The current frame
m_Delay		Used for game timing, the time till the next frame
m_State		User defined state. Walking, jumping, etc
m_Type		User defined type. Health, weapon, etc
	m_Angle	The sprite's angle of rotation
	m_Flipped	Is the sprite flipped?
m_Tile		A AGDXTile pointer to the sprite's bitmap data
m_Next		A AGDXSprite pointer to the next sprite in a AGDXSpriteList
m_Prev		A AGDXSprite pointer to the previous sprite in a AGDXSpriteList

المتغيرات (أو المؤشرات) أعضاء هذا العنصر :

موقع الممثل على المحور السيني (X)	m_PosX
موقع الممثل على المحور الصادي (Y)	m_PosY
تسارع الممثل على المحور السيني (X)	m_VelX
تسارع الممثل على المحور الصادي (Y)	m_VelY

صورة (Frame) الممثل الحالية	m_Frame
يستخدم هذا المتغير للتحكم في توقيت الحركة للممثل	m_Delay
يستخدم هذا المتغير لتحديد حالة الممثل (يمشي ، يقفز ... الخ)	m_State
يستخدم هذا المتغير لتحديد نوع الممثل (سلاح ، طاقة ... الخ)	m_Type
يستخدم هذا المتغير لتحديد زاوية دوران الممثل	m_Angle
يستخدم لمعرفة ما إذا كان الممثل يعرض بشكل مقلوب	m_Flipped
مؤشر عن طريق الكائن AGDXTile لصور الممثل	m_Tile
مؤشر للممثل التالي في القائمة التابعة للكائن AGDXSpriteList	m_Next
مؤشر للممثل السابق في القائمة التابعة للكائن AGDXSpriteList	m_Prev

Construction / Destruction الباني و المدمر

AGDXSprite(AGDXTile* pTile)

Creates a sprite object from a pointer to a AGDXTile object holding the sprite's bitmap data.

يقوم هذا الباني بإنشاء كائن للممثل من المؤشر للكائن AGDXTile والذي يحمل صور الممثل المطلوبة.

AGDXSprite(AGDXScreen *pScreen, char* szFilename, int w, int h, int num)

يقوم هذا الأمر (الباني الثاني) ويفرق بينه وبين الباني الأول عن طريق المعلومات الممررة) بإنشاء كائن للممثل ويحتاج إلى مؤشر للكائن AGDXScreen والموجود في كل برامج هذه المكتبة. المتغير w والمتغير h يعبران عن العرض والارتفاع لكل صورة للممثل والمتغير num يعبر عن مجموع الصور (frames) للممثل في ملف صورة واحد.

~AGDXSprite()

هذا المدمر ويقوم بتدمير كائن الممثل وتحرير الذاكرة المحجوزة له.

Member functions

الأوامر الأعضاء

void **Create**(AGDXTile* pTile)

عن طريق هذا الأمر ننشئ كائن للممثل من مؤشر للكائن AGDXTile والحامل لمعلومات عن الصور المطلوبة للمثل.

Create(AGDXScreen *pScreen, char* szFilename, int w, int h, int num)

يقوم هذا الأمر بإنشاء كائن للممثل ويحتاج لتميرير كائن الشاشة AGDXScreen واسم ملف الصورة المحتوي على صور الممثل. المتغير w و h يمثلان العرض والارتفاع لكل صورة للممثل والمتغير num يعبر عن مجموع الصور المختلفة للمثل في ملف الصورة (عندما نقول صور مختلفة نعني بها الصور frames التي تعبر عن حركة الممثل والتي تكون موجودة في صورة واحدة من نوع bmp)

BOOL **SpriteHit**(AGDXSprite* pSprite)

يقوم هذا الأمر بفحص التصادم بين الممثل وبين ممثل آخر وتكون النتيجة المرجعة أما TRUE أي صحيح عندما يكون جزء أو كل الممثل تصادم مع ممثل وخر أو تكون النتيجة المرجعة FALSE وتعني انه لم يحدث أي تصادم.

BOOL **TileHit**(AGDXMap* pMap, int Tile)

يقوم هذا الأمر بفحص التصادم بين الممثل وصورة تايل tile في خريطة التايلز وتكون النتيجة المرجعة أما TRUE أي صحيح عندما يكون جزء أو كل الممثل تصادم مع قيمة التايل (لكل

صورة تايل قيمة معينه تمثلها في خريطة التايلز ، راجع كائن الخلفيات المتحركة لمعلومات أكثر عن التايلز) أو تكون النتيجة المرجعة FALSE وتعني انه لم يحدث أي تصادم.

```
void SetPos(int pX, int pY) { m_PosX = pX; m_PosY = pY; }
void SetVel(int vX, int vY) { m_VelX = vX; m_VelY = vY; }
void SetFrame(int Frame) { m_Frame = Frame; }
void SetDelay(int Delay) { m_Delay = Delay; }
void SetState(int State) { m_State = State; }
void SetType(int Type) { m_Type = Type; }
```

هذه مجموعة الأوامر التي تستخدم لوضع المعلومات المطلوبة عن الممثل.

```
void SetVelocity(int xV, int yV)
```

عن طريق هذا الأمر نقوم بوضع التسارع المطلوب للممثل على المحورين x و y

```
void MoveTo(int x, int y)
```

عن طريق هذا الأمر نقوم بوضع الموقع المطلوب للممثل على المحورين x و y

```
HRESULT Draw(AGDXSurface* lpDDS)
HRESULT DrawFast(AGDXSurface* lpDDS)
HRESULT DrawTrans(AGDXSurface* lpDDS)
HRESULT DrawClipped(AGDXSurface* lpDDS)
HRESULT DrawWindowed(AGDXSurface* lpDDS);
HRESULT DrawScaled(float Factor, AGDXSurface* lpDDS)
void DrawRotated(float Angle, AGDXSurface* lpDDS)
void DrawHFlip(AGDXSurface* lpDDS)
void DrawVFlip(AGDXSurface* lpDDS)
```

تقوم كل هذه الأوامر بنسخ الصورة من السطح المصدر (وهو السطح المحتوي على الصورة) إلى السطح المشار إليه عن طريق المؤشر lpDDS. ويستخدم الصورة frame الحالية للممثل والأوامر m_PosX و m_PosY لتحديد موقع الصورة.

الاختلاف بين كل هذه الأوامر هو في طريقة النسخ وأسم كل أمر يدل على طريقته :

- ❖ Draw نسخ مباشر.
- ❖ DrawFast نسخ مباشر سريع.
- ❖ DrawTrans نسخ مع الأخذ في الاعتبار اللون المفتاح (الشفاف).
- ❖ DrawClipped نسخ مع الأخذ في الاعتبار أطراف الشاشة.
- ❖ DrawWindowed نسخ في مربع معين على الشاشة.
- ❖ DrawScaled نسخ مع تكبير أو تصغير الحجم.
- ❖ DrawRotated نسخ مع التدوير بدرجة معينة.
- ❖ DrawHFlip نسخ بشكل مقلوب أفقياً.
- ❖ DrawVFlip نسخ بشكل مقلوب رأسياً.

AGDXSpriteList

هذا الكائن يستخدم لتخزين الممثلين بشكل ديناميكي. وهو عبارة عن قائمة متصلة بحيث نستطيع أن نضيف ونحذف منها خلال عمل البرنامج.

LOOPING THE LIST : عندما نستخدم هذا الكائن سوف نحتاج إلى القيام بدورة على كل محتوياتها لنقوم بالعمليات المطلوبة لكل عضو بها كتحرك الممثل مثلا والبرنامج التالي يوضح كيفية القيام بهذه العملية :

```

AGDXSpriteList SpriteList;
AGDXSprite* Node;
AGDXSprite* Save;

for(Node = SpriteList.Next(); Node != SpriteList.List(); Node = Save)
{
    Save = Node->m_Next;

    //
    // تقوم بإضافة البرنامج المطلوب هنا لتغيير ما تشاء في الممثل
    //
}

لا حظ انك إذا أردت إزالة ممثل من القائمة خلال عمليه الدوران فيجب عليك أن تخزن
(باستخدام الأمر Save كما هو ظاهر في المثال ) موقع المؤشر للممثل التالي في القائمة بحيث لا
نفقد اتصال القائمة.
```

Data members

The top of the list	m_List
The number of sprites in the list	m_nSprites

المتغيرات (أو المؤشرات) أعضاء هذا العنصر :

أعلى القائمة	m_List
--------------	--------

m_nSprites مجموع عدد الممثلين في القائمة

Construction / Destruction الباني و المدمر

AGDXSpriteList(void)

~AGDXSpriteList(void)

Member functions

الأوامر الأعضاء

```
void AddSprite(AGDXTile* pTile, int Type=0,
                int PosX=0, int PosY=0,
                int VelX=0, int VelY=0,
                int State=0, int Frame=0,
                int Delay=0)
```

نستخدم هذا الأمر لإضافة ممثل إلى القائمة. وكل ما هو مطلوب مجرد مؤشر إلى الكائن AGDXTile ولكن تستطيع كذلك أن تقوم بأعداد الممثل كما تشاء (كإعطائه موقع معني مثلا).

```
void DelSprite(AGDXSprite* pNode)
```

نستخدم هذا الأمر لإزالة الممثل من القائمة المتصلة.

```
AGDXSprite* Next(void) { return m_List.m_Next; }
```

نستخدم هذا الأمر للحصول على مؤشر لأول ممثل في القائمة.

```
AGDXSprite* Prev(void) { return m_List.m_Prev; }
```

نستخدم هذا الأمر للحصول على مؤشر لآخر ممثل في القائمة.

```
AGDXSprite* List(void) { return &m_List; }
```

نستخدم هذا الأمر للحصول على مؤشر لأعلى ممثل في القائمة.

```
void Draw(AGDXSurface* lpDDS)
void DrawFast(AGDXSurface* lpDDS)
void DrawTrans(AGDXSurface* lpDDS)
void DrawClipped(AGDXSurface* lpDDS)
void DrawWindowed(AGDXSurface* lpDDS)
void DrawScaled(float Factor, AGDXSurface* lpDDS)
void DrawRotated(float Angle, AGDXSurface* lpDDS)
void DrawHFlip(AGDXSurface* lpDDS)
void DrawVFlip(AGDXSurface* lpDDS)
```

تقوم كل هذه الأوامر بنسخ الصورة من السطح المصدر (وهو السطح المحتوي على الصورة) إلى

السطح المشار إليه عن طريق المؤشر lpDDS.

الاختلاف بين كل هذه الأوامر هو في طريقة النسخ وأسم كل أمر يدل على طريقته:

- ❖ Draw نسخ مباشر.
- ❖ DrawFast نسخ مباشر سريع.
- ❖ DrawTrans نسخ مع الأخذ في الاعتبار اللون المفتاح (الشفاف).
- ❖ DrawClipped نسخ مع الأخذ في الاعتبار أطراف الشاشة.
- ❖ DrawWindowed نسخ في مربع معين على الشاشة.
- ❖ DrawScaled نسخ مع تكبير أو تصغير الحجم.
- ❖ DrawRotated نسخ مع التدوير بدرجة معينة.
- ❖ DrawHFlip نسخ بشكل مقلوب أفقياً.
- ❖ DrawVFlip نسخ بشكل مقلوب رأسياً.

AGDXInput

AGDXInput هو الكائن المسؤول عن الأوامر المطلوبة للإدخال ويستطيع التعامل مع الفأرة (mouse) و لوحة المفاتيح (keyboard) وكذلك عصا الألعاب (joystick) في الوقت الحاضر يتعامل هذا الكائن مع الإحداثيين السيني والصادي للفأرة بشكل نسبي وليس بشكل مطلق . بمعنى انه سيقوم بإضافة نقطه لكل وحدة زمنية في الإحداثي المطلوب أما إذا أردت أن تحصل على الإحداثي المطلق بحيث تعرف موقع إشارة الفأرة بالنسبة للشاشة فيجب أن تستخدم الكائن m_lpDIDMouse.

Data members

The DirectInput object	m_lpDI
The DirectInputDevice for the keyboard	m_lpDIDKeyboard
The DirectInputDevice for the mouse	m_lpDIDMouse
A POINT structure holding the relative mouse position	Mouse
Joystick	A POINT structure holding the joystick X and Y positions
Is the left button pressed?	MouseLB
Is the middle button pressed?	MouseMB
Is the right button pressed?	MouseRB
Is the joystick button 1 pressed?	JoystickB1
Is the joystick button 2 pressed?	JoystickB2
A 256 array of BYTE's, stores the current key presses	Keys
A JOYINFOEX structure holding joystick information	JoyInfo

المتغيرات (أو المؤشرات) أعضاء هذا العنصر :

كائن DirectInput	m_lpDI
كائن DirectInputDevice للوحة المفاتيح	m_lpDIDKeyboard
كائن DirectInputDevice لإشارة الفأرة	m_lpDIDMouse
متغير من نوع struct يحمل موقع إشارة الفأرة	Mouse
متغير من نوع struct يحمل موقع (x,y) عصا الألعاب	Joystick
هل الزر اليسار لإشارة الفأرة تم ضغطة	MouseLB

هل الزر الوسط لإشارة الفأرة تم ضغطة	MouseMB
هل الزر اليمين لإشارة الفأرة تم ضغطة	MouseRB
هل الزر الأول لعصا الألعاب تم ضغطة	JoystickB1
هل الزر الثاني لعصا الألعاب تم ضغطة	JoystickB2
متغير من نوع شعاع (array) يحمل المفتاح الذي تم ضغطة	Keys
متغير من نوع struct يحمل معلومات عن عصا الألعاب	JoyInfo

Construction / Destruction الباني و المدمر

AGDXInput()
~AGDXInput()

Member functions

الأوامر الأعضاء

BOOL Create(void *hInst, void *hWnd)
Creates the DirectInput object and the keyboard, mouse and joystick devices.

يقوم هذا الأمر بإنشاء كائن الإدخال DirectInput وكائن للوحة المفاتيح وكائن للفأرة وأخير كائن لعصى الألعاب.

void Restore(void)
Requires the keyboard and mouse devices if lost. Called internally if **Update** fails.

عن طريق هذا الأمر نستطيع الحصول على كائن إشارة الفأرة ولوحة المفاتيح في حالة ضياعها (تضيق إذا حول المستخدم التركيز عن البرنامج) وينفذ هذا الأمر بشكل تلقائي من المكتبة في حالة لم تستطع تنفيذ أمر التجديد **Update** (انظر في الأسفل)

void Release(void)
Releases the keyboard and mouse devices.

عن طريق هذا الأمر نحرر الذاكرة المحجوزة لكائن الفأرة ولوحة المفاتيح.

void Update(void)

Updates the keyboard, mouse and joystick data.

هذا أمر التجديد ويقوم بتجديد المعلومات عن لوحة المفاتيح وإشارة الفأرة وعصا الألعاب (لأنها في تغيير مستمر).

AGDXMusic

كائن الموسيقى يسمح لك بتعبئة ولعب الملفات الموسيقية من نوع MIDI في برامجك. حالياً هذا الكائن يستخدم أوامر الـ MCI للقيام بعمله.

Data Members

Window handle HWND m_hWnd

المتغيرات (أو المؤشرات) أعضاء هذا العنصر :

HWND m_hWnd هذا المتغير الحامل لعنوان البرنامج.

Construction / Destruction

AGDXMusic(void *hWnd)
Creates a AGDXMusic object. Assigns m_hWnd to hWnd.

هذا الأمر الباني ويقوم بأخذ المتغير hWnd الحامل لعنوان البرنامج لكي يعطي قيمته للمتغير
m_hWnd

Member functions

الأوامر الأعضاء

BOOL Play(char *Filename)
يقوم هذا الأمر ببدء لعب ملف MIDI المطلوب

BOOL Stop(void)

يقوم هذا الأمر بإيقاف لعب ملف MIDI المستعمل حالياً

BOOL Pause(void)

يقوم هذا الأمر بإيقاف مؤقت في لعب ملف MIDI المستعمل حالياً.

BOOL Resume(void)

يقوم هذا الأمر بلعب ملف MIDI الموقوف بشكل مؤقت عن طريق الأمر السابق.

BOOL Restart(void)

يقوم هذا الأمر بإعادة لعب ملف MIDI المستعمل حالياً من البداية.

AGDXSound

كائن الصوت يقوم بتبسيط التعامل مع العنصر DirectSound من عناصر تكنولوجيا DirectX ونقوم أولاً بإنشاء هذا العنصر قبل تعبئة الأصوات ولعبها باستعمال كائن AGDXSoundBuffer (أنظر إلى الكائن القادم).

Data members

The DirectSound Object	LPDIRECTSOUND m_lpDS
Direct sound capabilities	DSCAPS m_DSCaps

المتغيرات (أو المؤشرات) أعضاء هذا العنصر :

المؤشر الممثل لكائن الصوت DirectSound	LPDIRECTSOUND m_lpDS
المتغير المحتوي على إماكنيات كائن الصوت	DSCAPS m_DSCaps

Construction / Destruction الباني و المدمر

AGDXSound()
~AGDXSound()

Member functions

الأوامر الأعضاء

HRESULT **Create**(void *hWnd)

يقوم هذا الكائن بإنشاء كائن الصوت DirectSound. بالنسبة للقيمة المرجعة أما أن تكون نوع الخطأ الذي حصل عند إنشاء هذا الكائن عن طريق الأمر SetCooperativeLevel أو القيمة DS_OK والذي يعني أن كل شيء على ما يرام.

HRESULT GetCaps(void)

عن طريق هذا الأمر نستطيع الحصول على إمكانيات الصوت الموجودة في الجهاز. بالنسبة للقيمة المرجعة فإنها تحتوي على نوع الخطأ الذي حصل عن طريق الأمر GetCaps أو القيمة DS_OK والذي يعني أن كل شيء على ما يرام.

AGDXSoundBuffer

هذا هو الكائن المسؤول عن تعبئة ولعب الملفات الصوتية من نوع WAVE والتي تستخدم لإعطاء المؤثرات الصوتية في البرنامج.

Data Members

int m_nBuffers	The number of sound buffers created
int m_Current	The currently selected buffer
const char* m_Filename	The name of the WAVE file
AGDXSound* m_pDS	Contains a pointer to the AGDXSound object
The IDirectSoundBuffer object	LPDIRECTSOUNDBUFFER m_lpDSB[1]

المتغيرات (أو المؤشرات) أعضاء هذا العنصر :

عدد نسخ الصوت التي تم إنشائها	int m_nBuffers
النسخة المختارة حاليا	int m_Current
اسم الملف من نوع WAVE للصوت	const char* m_Filename
مؤشر للكائن	AGDXSound* m_pDS
	AGDXSound
مؤشر لهذا الكائن	LPDIRECTSOUNDBUFFER m_lpDSB[1]

Construction / Destruction الباني و المدمر

AGDXSoundBuffer(void)
~**AGDXSoundBuffer**(void)

Member functions

الأوامر الأعضاء

BOOL Load(AGDXSound* pDS, char *Filename, int Num = 1)

يقوم هذا الأمر بتعبئة الصوت WAVE من الملف المشار إليه عن طريق المتغير Filename. أما المتغير Num فيدل على عدد النسخ المحتمل أن يتم لعبها في وقت واحد. (مثلا إذ كنت تريد استخدام صوت انفجار وتتوقع استخدام صوت الانفجار أحيانا خمس مرات في لحظة واحدة تأكد أن تضع هذا المتغير يساوي 5) ولكن حاول أن تبقي هذا الرقم صغيرا قدر الإمكان.

BOOL Play(int Pan = 0, DWORD dwFlags = 0)

عن طريق هذا الأمر تستطيع لعب ملفات WAVE. وإذا كان الجهاز يحتوي على بطاقة صوت ستيريو (جميع البطاقات حاليا في الأسواق لديها قدرة ستيريو) نستطيع أن نتحكم في الصوت من السماعه اليمين أو السماعه اليسار عن طريق إعطاء المتغير Pan القيمة المطلوبة بحيث أن القيمة 10,000 هي أقصى اليمين والقيمة -10,000 هي أقصى اليسار والقيمة صفر (وهي القيمة التي تختارها المكتبة لنا إذا لم نقم بأي تغييرين) يكون الصوت متساوي من الجهتين. ونستطيع كذلك عن طريق المتغير dwFlags التحكم في تكرار الصوت عن طريق إعطاءه القيمة DSBPLAY_LOOPING عندها سوف يتكرر لعب الصوت كلما وصل إلى نهايته.

BOOL Stop()

يقوم هذا الأمر بإيقاف لعب الملف الصوتي.

void SetVolume(LONG Volume)

نستطيع عن طريق هذا الأمر التحكم في حدة الصوت. فعن طريق تغيير قيمة المتغير Volume بحيث أن القيمة صفر تدل على أن الصوت لن يسمع والقيمة 10,000 تدل على أن الصوت سوف يسمع بالشكل الذي سجل عليه ونستطيع التحكم في القيم كما نشاء بين هذين الرقمين (لا نستطيع تكنولوجيا DirectX رفع الصوت لذلك فإن أي قيمة أعلى من 10,000 لن تقدم أي تغيير)

BOOL **CreateSoundBuffer**(DWORD dwBufSize, DWORD dwFreq,
DWORD dwBitsPerSample, DWORD dwBlkAlign, BOOL bStereo)

BOOL **ReadData**(FILE* fp, DWORD dwSize, DWORD dwPos)

كلا هذين الأمرين يتم استدعاؤهما عن طريق المكتبة بشكل تلقائي عن طريق الأمر LoadFromFile. وليس من الضروري استخدامهما بشكل مباشر.

Global Functions

هذه الأوامر العامة التي تستطيع استخدامها في أي جزء من برنامجك ، وهي مفيدة جدا لمساعدتك في تطوير البرنامج .

```
void DDError(HRESULT hErr, void * hWnd)
void DSError(HRESULT hErr, void * hWnd)
```

هذين أمرين مفيدتين لإيجاد نوع الخطأ إذا حصل ويقومان بعرض الخطأ في مربع حوار خاص يطلق عليه MessageBox استخدمهما في الحالات الضرورية.

```
void Clip(int *DestX, int *DestY, RECT *SrcRect, RECT DestRect)
```

عن طريق هذا الأمر نستطيع أن نحدد منطقة مستطيلة معينة على الشاشة بحيث تكون شاشة بحد ذاتها ونستطيع من خلالها أن نعرض ما نشاء وتكون زوايا المنطقة المستطيلة مثل زوايا الشاشة في حالة لو أن الممثل أو الخلفية خرجت عن حدودها فإنها سوف تختفي. اعتقد انك تستطيع تصور الكثير من الاستخدامات لهذا الأمر أفرض على سبيل المثال انك تبرمج لعبة شطرنج وتريد أن تكون المنطقة اليسار مثلا 100 نقطة من اليسار مخصصة لإعطاء المعلومات المطلوبة للاعبين وبقيّة الشاشة تكون للعبة نفسها. هذا الأمر كذلك يتم طلبه عن طريق أمر الرسم DrawClipped الذي تعرفنا عليه سابقا.

ملاحظات مهمة :

- يجب استخدام أمر للون الشفاف قبل القيام بعملية الرسم للسطح حتى نستطيع استخدام هذه الميزة (مثلا أمر الرسم DrawTrans لكائن المثلين أو لكائن الخريطة يجب استخدامه بعد أن نكون قد استخدمنا أمر اللون الشفاف ColorKey() لإعطاء للون الشفاف المطلوب)

- الأمر `m_lpclipper` (أحد أعضاء كائن الشاشة `AGDXScreen`) ليس للاستخدام في حالة لو أن البرنامج يستغل الشاشة كاملة.
- عندما تكون ذاكرة الفيديو غير كافية لحمل المعلومات المعينة للبرنامج فإن المكتبة سوف تقوم باستغلال ما على الجهاز من الذاكرة (وتسمى ذاكرة النظام وتختلف عن ذاكرة الفيديو) مما ينتج عنه بطئ عام في سرعة البرنامج وهذا شيء طبيعي بما أن ذاكرة الفيديو أسرع بكثير من ذاكرة النظام بالنسبة للرسوم على الشاشة. لذلك يجب أن تكون حذرا من عدد الأسطح التي تنشئها.
- البعد النقطي `320x200` و `320x240` ليس جيد للخلفيات المتحركة وذلك راجع لطبيعتهما لأنهما يعرفان بـ `ModeX` وطبيعتهما المعروفة لمبرمجي الألعاب على النظام دوس.
- الأمر `DrawWindowed` يستعمل للرسم بشكل مباشر للسطح الرئيسي على الشاشة.
- العضو `DestRect` أحد أعضاء كائن الأسطح `AGDXSurface` يستخدم فقط لأوامر الرسم.
- عندما نقوم بإنشاء الخلفيات المتحركة في نافذة من نوافذ `Windows` يجب أن نتأكد أن حجم الشاشة هو نفس الحجم النقطي الذي أنشأناه في أمر كائن الشاشة للبرنامج (يعني يجب أن لا نصغر أو نكبر النافذة حتى تكون حركة الخلفية سلسة)

لحظة , لازل عندي سؤال !!!!



عزيزي القاري ليست هذه نهاية المطاف بل بدايته , أتمنى أن تكون قد استفدت من مواضيع هذا الكتاب ولا بد أن يكون لديك بعض الأسئلة. وهذا شيء طبيعي , فإذا حصل لك ذلك في موضوع معين فأنصحك أن تعيد قراءته مرة أخرى. لأنني عند كتابة كل موضوع في هذا الكتاب كنت اكتبه ثم أقرئه عدة مرات , بعدها أسأل نفسي هل هو واضح أم لا. فإذا وجدت أن هناك بعض الغموض في جزء معين قمت بإعادة كتابته. ولكن مع ذلك فأنا متأكد أن هناك بعض الأجزاء التي ولا بد سيكون لك بعض التساؤل بها. ولهذا السبب فأنا أشجعك عزيز القارئ على إرسال رسالة إلكترونية لي عن طريق بريدي الإلكتروني (انظر في الصفحة المقابلة). وأنا منذ كتابة الكتاب الأول لم اذكر أنني أهملت أو تركت أي تسأل من قارئ بدون إجابة. وسوف أحاول الرد عليك في أسرع وقت ممكن. كما أنني أنصحك عزيزي

القاري بزيارة صفحتي على الإنترنت من وقت لآخر لأنني أضع بها أي إضافات أو معلومات أجدها مهمة. كما أن هناك ساحة للنقاش تستطيع أن تطرح أسئلتك بها. وتذكر لا يوجد شي اسمه "سؤال تافه" لان كل الأسئلة مهمة مهما كانت.

المهندس : عبدالناصر الكعبي

البريد الإلكتروني :

trytofindme@hotmail.com
alkaabi@arabgames.com

إذا لم تجد رد علي أي منها خلال مدة 48 ساعة استخدم عنوان مختلف.

الموقع على الإنترنت :

<http://www.ArabGames.com>

هذا الموقع ليس فقط لإجابة أسئلة هذا الكتاب ولكن كذلك للالتقاء بالآخرين ممن لهم نفس الميول وتستطيع أن تستفيد أو أن تشارك خبرات الآخرين.

برمجة الأبعاد الثنائية والثلاثية للكمبيوتر

تعتبر برمجة الصوت والصورة **Multimedia** من أصعب أنواع البرمجة على الحاسب الآلي ولكنها أيضا أكثرها إثارة. أكتشف كيف تستطيع أن تجعل من برمجة الألعاب ،التصور الحقيقي ، الأبعاد الثنائية والثلاثية باستخدام تكنولوجيا **DirectX** عملاً سهلاً وممتعاً. تعرف على هذه التكنولوجيا بشكل بسيط ومفصل وأكتب برنامجك الأول ليس في أسابيع أو أيام أو حتى ساعات بل في دقائق معدودة عن طريق أمثلة الكتاب الكثيرة.

أغلب البرامج التي صممت للأنظمة **Windows 95/98** و **Windows 2000** تستغل تكنولوجيا **DirectX** الذهلة من مايكروسوفت. تعرف على هذه التكنولوجيا وكيفية التعامل معها. تعرف على كيفية إنشاء ألعاب الكمبيوتر والبرامج ثلاثية الأبعاد والتي يستخدمها الملايين في أنحاء المعمورة. لم ينس الكاتب أن الكثير من القراء قد يكونوا من الهواة أو المبتدئين أو متمرسين في لغات الكمبيوتر لذلك ينقسم هذا الكتاب إلى قسمين : الأول للمبتدئين ويستخدم مكتبة **AGDX** لإنتاج برامج الصوت والصورة بشكل سريع وسهل ولكن ففعال. أما الجزء الثاني فهو موجهة للمتمرسين وفيه سوف ننظر بقرب إلى طريقة عمل تكنولوجيا **DirectX** وكيفية التعامل معها. هناك الكثير من الأمثلة في هذا الكتاب لإجابة أغلب الأسئلة التي قد يطرحها القارئ في هذا الموضوع.

تأليف المهندس : عبدالناصر سيف الكعبي



أكتشف كيف تستطيع باستخدام البرنامج **Bryce** إنتاج مناظر طبيعية ثلاثية الأبعاد واستخدامها في برامجك. وأنظر كيف قمنا بذلك مع أمثلة الكتاب الكثيرة. (النسخة التجريبية من البرنامج ملحق مع الكتاب)



أصعب ما يواجه المبرمجين الجدد هو إنتاج الرسوم ثلاثية الأبعاد المتحركة للكائنات الحية ،سواء للإنسان أو للحيوان. تعرف على البرنامج **Poser** ونظر كيف جعلنا من إنشائها وتحريكها عمل سهل بل وممتع.

