**بسم الله الرحمن الرحيم**

[**مشروع تعليميّ لـ Direct3D**](file:///G:\advanced\6-%20للمحترفين%20فقط\DirectX.htm)

**باستخدام VB.Net**

**إعداد:**

**محمد حمدي غانم**

**تأسيسا على ترجمة متصرّفة، لكتاب:**

**Basic Direct3D 9.0**

**The Managed Way**

**لكاتبه:**

**Jack Hoxley**

**من موقع:**

**http://www.DirectX4VB.com**

**ملاحظتان:**

**1- هذا الموضوع متقدّم جدّا، ويتطلّب منك إجادة قواعد البرمجة أولا.. فلا تحاول قراءة هذا الفصل قبل إجادة الفصول السابقة له.**

**2- أرجو مراعاة أنّ خبرتي في DirectX9 ما زالت متواضعة، ولا تتجاوز المعلومات الموجودة في هذا المشروع (!!)، فمن أن أراد أن يراسلني فلا يعتقد أنّه يتعامل مع علاّمة:**

[**msvbnet@hotmail.com**](mailto:msvbnet@hotmail.com)

**ويمكنكم مراسلة (جاك هوكسلي) صاحب الموقع والكتاب والمشروع، على العنوان التالي:**

**Jack.Hoxley@DirectX4VB.com**

**مع تحيّاتي، وتمنّياتي بالتوفيق:**

**محمد حمدي غانم**

**مشروع تعليميّ لـ Direct3D**

* [**مقدمة حول Microsoft DirectX 9.0:**](file:///G:\advanced\DirectX\1-%20مقدمة.htm)
* [**مفاهيم أساسيّة:**](file:///G:\advanced\DirectX\2-%20مفاهيم%20أساسيّة.htm)
* [**استخدام Direct3D9:**](file:///G:\advanced\DirectX\3-%20استخدام.htm)
* [**مشروع تعليميّ:**](file:///G:\advanced\DirectX\4-%20مشروع%20تعليميّ.htm)

**مقدمة حول Microsoft DirectX 9.0:**

منذ فترة، وميكروسوفت معنيّة بتطوير أدواتها الخاصّة بالرسوم والألعاب والوسائط المتعدّدة Multimedia.. وفي هذا الصدد، قامت ميكروسوفت بتطوير ما أسمته DirectX.. ويعنينا هنا أن نتوقّف قليلا أمام هذه التسميّة.

أمّا بخصوص كلمة "مباشر" Direct، فهي ناتجة من أنّ هذه التقنية تستخدم مجموعة منخفضة المستوى Low Level من واجهات برمجة التطبيقات Application Programming Interfaces (APIs)، التي تتيح لمن يستخدمها أن يتعامل ((**مباشرةً**)) مع مكونات الكمبيوتر، بدون وسيط من تقنية COM، ممّا يضمن سرعةً وكفاءةً وتحكّما أعلى في تنفيذ التطبيقات التي تستخدم هذه الوسيلة.

وقبلَ أن ينقبضَ صدرك مع سماعة لتعبيرات Low Level و API و COM، يجب أن أخبركَ أنّ شيئا لن يختلف بالنسبة لك كمبرمج VB، فقد قامت ميكروسوفت بإنتاج نسخة من DirectX في صورة خلايا Classes، بحيث يمكنك استخدامها من داخل VB بنفس الطريقة التي تستخدم بها باقي خلايا إطار العمل Framework.. تنفّس الصُعَداء إذن.

هذا عن كلمة Direct.. ماذا إذن عن هذا الحرف X، الذي يشغف الأمريكان حبّا؟!

إنّ هذا الحرف يقوم عوضا عن أيّ مجهول، يمكن أن يتّخذ مجموعة مختلفة من القيم.. إنّ هذا إذن يعني أنّ DirectX يحتوي على مجموعة مختلفة من الأدوات، مثل Direct3D و DirectDraw و DirectInput.

إنَ حوالي 75% على الأقّل من الألعاب تحتاج لإصدار من إصدارات DirectX المختلفة لكي تعمل.. إنّ هذا يشي بوضوح بأهمّيّة هذه التقنية، ومدى الإمكانيّات التي تقدّمها.

**مفاهيم أساسيّة**

* [**عالم المجسّمات:**](file:///G:\advanced\2-%20مفاهيم%20أساسيّة\1-%20عالم%20المجسّمات.htm)
* [**نظام الإحداثيات ثلاثي الأبعاد 3-D Coordinate Systems:**](file:///G:\advanced\2-%20مفاهيم%20أساسيّة\2-%20نظام%20الإحداثيات.htm)
* [**الرءوس Vertices:**](file:///G:\advanced\2-%20مفاهيم%20أساسيّة\3-%20الرءوس.htm)
* [**الخاماتTextures :**](file:///G:\advanced\2-%20مفاهيم%20أساسيّة\4-%20الخامات.htm)
* [**التحويلات Transformations:**](file:///G:\advanced\2-%20مفاهيم%20أساسيّة\5-%20التحويلات.htm)
* [**تحويل الإحداثيّات World Ttransform:**](file:///G:\advanced\2-%20مفاهيم%20أساسيّة\6-%20تحويل%20الإحداثيّات.htm)
* [**تحويلات الكاميرا Camera Transforms:**](file:///G:\advanced\2-%20مفاهيم%20أساسيّة\7-%20تحويلات%20الكاميرا.htm)
* [**تحويلات المنظور Projection Transforms:**](file:///G:\advanced\2-%20مفاهيم%20أساسيّة\8-%20تحويلات%20المنظور.htm)
* [**العناصر الهندسيّة Meshes والنماذج Models:**](file:///G:\advanced\2-%20مفاهيم%20أساسيّة\9-%20العناصر%20الهندسيّة.htm)

**عالم المجسّمات:**

اكتسبَ الكمبيوتر في الفترةِ الأخيرة قوّةً هائلة، حتّى لقد صار بإمكانك أن تصمّم وتعرض الرسوم المجسّمة على جهازك الشخصيّ.. حتّى لقد وصل الأمر إلى إنتاج أفلام كاملة بالرسوم المجسّمة، التي تذهل من يشاهدها من فرط مقاربة تفاصيلها من الواقع.

ولكن لا تظنّ الأمر تافها، فما زالت عمليّة تكوين المجسّمات على أجهزة الكمبيوتر Rendering بطيئة، نظرا للكمّ الهائل من العمليّات التي يتمّ إجراؤها، حتّى إنّ تكوين صورة مجسّمة واحدة (أي تحويلها من طور التصميم إلى ملفّ فيديو)، قد يستغرق عدّة دقائق ـ على أحسن الفروض.

طبعا ستقول في نفسك مستخفّا: وماذا في عدّة دقائق؟

ولكي تعرف الإجابة، يجب أن أذكّرك أنّ الرسوم المتحرّكة عبارة عن مجموعة هائلة من الصور، يتمّ عرضها متتابعة بسرعة (أكثر من 20 لقطة في الثانية)، بحيث تنخدع عين الرائي فترى الرسوم تتحرّك.. هل تخيّلت إذن كم صورة في دقيقة واحدة؟.. أكثر من 1200 صورة.. حاول أن تتخيّل إذن عدد الساعات الذي ستستهلكه عمليّة تكوين مشهد رسوم مجسّمة مدّته دقيقة واحدة!

ولكن لا تجزع.. إنّ التطوّر السريع لإمكانيات الكمبيوتر سيجعل هذا الوقت يتقلّص باستمرار.

ثمّ إنّنا ـ كمبرمجين ـ لن نقوم بهذه العمليّة.. إنّ كلّ ما سنفعله هو استلام المجسّمات من مصمّميها (بالاستعانة بتطبيقات مثل 3DS Max و TrueSpace و Lightwave و Maya وغيرها)، واستخدامها في تطبيقاتنا كما يحلو لنا.

وفي هذا الفصل، سنركّز جهودنا لتعلّم القدرات المدهشة التي يمنحها لنا DirectX عبر تقنية Direct3D.

إنّ Direct3D يقف بيننا وبين كارت الشاشة مباشرةً، وذلك منعا لإضاعة أيّ وقت، فكما ذكرنا سابقا، تستهلك الرسوم المجسّمة وقتا ملموسا بالفعل.

ملحوظة1:

يجب أن يدعم كارت الشاشة الخاص بك الرسوم المجسّمة، وإلا فلن تعمل التطبيقات التي ستنشئها باستخدام Direct3D.

ملحوظة2:

لا علاقة لك كمبرمج بنوع مكوّنات الجهاز الماديّة Hardware، فهي لن تؤثّر علي الكود الذي تكتبه.. إنّ Direct3D سيتّولى عنك هذه المسئوليّة، فهو يعرف كيف يتحاور مع أنواع المكوّنات المختلفة.

**نظام الإحداثيات ثلاثي الأبعاد 3-D Coordinate Systems:**

تستخدم الرسوم المجسّمة نوعين من نظم الإحداثيّات: اليساريّ Left-handed واليميني Right-handed.. وفي كلا النظامين، يشير محور السينات الموجب Positive x-axis إلى اليمين، ومحور الصادات الموجب Positive y-axis لأعلى.

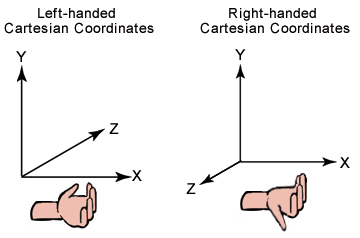
ملحوظة:

انتبه لهذه الاتجاهات جيّدا، ولا ترتبك بينها وبين إحداثيّات الشاشة في النظام ثنائيّ البعد 2D.. (تذكّر أنّ محور الصادات الموجب Positive y-axis في VB يشير لأسفل).

في أيّ اتجاه إذن يشير محور العينات الموجب Positive z-axis؟

بسيطة.. كلّ ما عليك هو أن تطبّق قاعدة اليد اليسرى في النظام اليساريّ، وقاعدة اليد اليمنى في النظام اليمينيّ!

ولا تمتعض هكذا حال سماعك كلمة قاعدة، فالموضوع في غاية السهولة:



اجعل أصابعك الثلاثة الإبهام والسبّابة والوسطى متعامدة، واجعل سبّابتك تشير إلى اتجاه محور السينات الموجب، ووسطاك تشير إلى اتجاه محور الصادات الموجب.. في هذه الحالة سيشير إبهامك إلى اتجاه محور العينات الموجب.

وطبعا في نظام الإحداثيّات اليمينيّ يجب أن تستخدم أصابع يدك اليمني، وفي نظام الإحداثيّات اليساريّ يجب أن تستخدم يدك اليسرى.

وكما هو واضح من الرسم، يشير اتجاه محور العينات الموجب إلى خارج الشاشة في نظام الإحداثيّات اليمينيّ، وإلى داخل الشاشة في نظام الإحداثيّات اليساريّ.

هذا، ويستخدم Direct3Dنظام الإحداثيّات اليساريّ Left-handed coordinate system.

طبعا ستتساءل في ضجر، عن سبب كلّ هذه الثرثرة.. إنّ هذا يعود لسبب جوهريّ.. هو أنّك لن تستخدم Direct3D بمفرده، فلا بدّ أنّك ستستعين ببعض تطبيقات الرسوم ثلاثّية الأبعاد الشهيرة، لتصميم الشكل الذي تريده ـ وذلك للاستفادة من التسهيلات التي تقدّمها لك هذه التطبيقات.. في هذه الحالة قد تصطدم بأنّ بعض هذه التطبيقات يستخدم نظام الإحداثيّات اليمينيّ!!... فماذا ستفعل إذن يا ترى في هذه الحالة؟

طبعا يجب عليك أن تحوّل من النظام اليمينيّ إلى النظام اليساريّ.. عامّة لا تقلق.. هناك من الدوال ما سيساعدك على القيام بهذا.

**الرءوس Vertices:**

لا ريب أنّك تتساءل عن كيفيّة تمثيل عالم ثلاثيّ الأبعاد على شاشة مسطّحة؟!

إنّ بإمكانك تخيّل ذلك، لو علمت أنّ كلّ شيء نرسمه على الشاشة ما هو في النهاية إلا مثلثا.. نعم.. كلّ الأشكال والتقاطعات يمكن تمثيلها على شاشة الكمبيوتر بمجموعة من المثلّثات.

ولكن لماذا المثلّث بالذات؟

أولا: لأنّه شكل مغلق.

ثانيا: لأنّه يتكوّن من أقلّ عدد من الرءوس.

ثالثا: وهو الأهمّ، أنّنا نضمن دائما أنّ رءوسه موجودة في مستوى واحد، فمن الممكن عند توصيل أربعة نقاط أو أكثر ألا تشكّل شكلا مستويا، لأنّ نقطةً أو أكثر قد تكون موجودة في المستوى العموديّ على مستوى باقي النقاط.

وطبعا لن نتدخّل نحن في تمثيل المجسّمات على الشاشة المسطّحة، فهناك معادلات رياضيّة معقّدة تقوم بتحويل إحداثيّات الأشكال الخاصّة بنا من النظام ثلاثيّ البعد إلى النظام ثنائيّ البعد، دون أن تفقد العين إحساسها بالتجسيم.

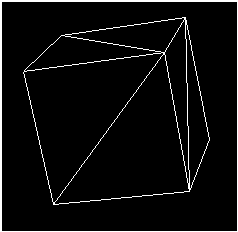
ولا تجعل الأمر يدهشك.. إنّ الشاشة تجبرك دائما على أن تنظر للشكل من زاوية واحدة (فأنت لا تتوقّع أن تدور حول الشاشة لتشاهد الشكل من خلفيّته!!).. إنّ هذا معناه أنّ بإمكاننا أن نريك صورة ثنائيّة البعد تمثّل الشكل من هذه الزاوية، مع إحساس بوجود العمق.. فإذا كان الشكل يتحرّك، فإنّ معنى هذا أنّ الزاوية التي نراه منها ستختلف.. هنا سيقوم Direct3D بإجراء المعادلات الرياضيّة اللازمة لرسم صورة جديدة للشكل تبرزه من هذه الزاوية.. وبهذا ستشعر أنت أنّك في عالم مجسّم حقيقيّ، لأنّ كلّ التفاصيل التي تريدها موجودة!

إنّ كلّ ما نريد أن نعرفه هنا، هو كيفيّة تمثيل الأشكال المجسّمة.

طبعا أنت تعرف أنّ للمثلّث ثلاثة رءوس 3 vertices.

إنّ الرأس vertex هو أبسط وأصغر وحدة سنستخدمها للتعبير عن البيانات في الرسوم المجسّدة.. هذا الرأس ما هو إلا نقطة في النظام ثلاثيّ الأبعاد، ممّا يعني أنّ تعريفها يحتاج لمعرفة إحداثيّاتها الثلاثة: (س، ص، ع).. ولكنّ الإحداثيّات ليست هي كلّ المعلومات التي تخصّ النقطة.. هناك أيضا لونها، درجة إضاءتها، خامتها.. إلخ.. وكلّها عوامل تؤثّر في سلوك هذه النقطة عند تحرّكها وتفاعلها مع ما حولها، كما سنرى فيما بعد.

تعال نرى مثالا لكيفيّة تمثيل أحد المجسّمات، وليكن المكعّب:



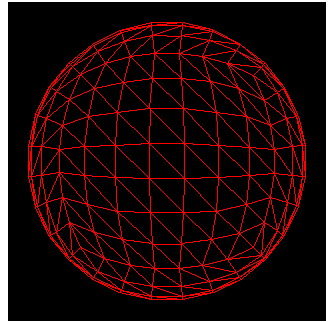
إنّ للمكعّب ستّةَ أوجه، كلّ منها عبارة عن مربّع.. وأنت تعرف أنّ قطر المربّع يقسمه لمثلّثين متماثلين.. معنى هذا أنّ المكعّب يتكوّن من 12 مثلّثا.

هل يعني هذا أنّنا نحتاج لـ 3 × 12 = 36 رأسا لتمثيل المكعّب؟

لا بالطبع!.. إنّ كلّ مثلثين متجاورين يشتركان في أحد الأضلاع (أيّ يشتركان في رأسين!)..

إنّنا إذن نحتاج لـ 8 رءوس فقط، حيث يمكن الحصول على كلّ المثلثات بتوصيل كلّ ثلاثة من هذه الرءوس معا.

أعرف أنّ الشكّ سيراودك حول قدرة المثلّثات حلى تمثيل كلّ المجسّمات.. الشكل التالي سيقنعك أنّ المثلثات قادرة حتّى على تمثيل الأسطح المحدّبة والمنحنية:



**الخامات Textures:**

لن يكون عالمك المجسّم قريبا من الحقيقة، ما لم تراعِ نوعيّة الخامات الداخلة في تكوين عناصره.. وإلا فكيف يمكنك مثلا رسم فقّاعة صابون تعلو في الفضاء.. إنّ هذه الفقّاعة ستعكس بعض المشاهد، وستشفّ عن بعض المشاهد في خلفيّتها.. إنّ من العبث أن تعتقد أنّك ستبرمج كلّ ذلك!

كلّ ما عليك فعله، هو أن تستعين بأحد محترفي الرسوم المجسّمة ليعطيك صورة مجسّمة للفقاعة والمكان من حولها.. ثمّ باستخدام Direct3D يمكنك تحديد خامات كلّ مادة ودرجة شفافيتها ولونها وإضاءتها... إلخ...

وعندما تقوم بتحريك الفقاعة، سيتولّى Direct3D إحداث التأثيرات التي تشعرك بأنّ المشهد حقيقيّ تماما، وذلك عبر خوارزميّات معقّدة للغاية بالطبع.

ولكن كيف يتمّ تمثيل الخامات؟

إنّ الخامة هي صورة ثنائيّة البعد (مخزّنة كمصفوفة نقاط Bitmap) يتمّ لصقها على مثلّث أو مجموعة من المثلثات، لإعطاء جزء من الشكل المجسّم التأثير المطلوب.

وسنتعرّف على ذلك بالتفصيل فيما بعد.

**التحويلات Transformations:**

قلنا من قبل إنّ المجسّمات التي نمثّلها في الإحداثيّات ثلاثيّة البعد، يجب أن يتمّ تحويلها إلى رسوم ثنائيّة البعد حتّى يمكن عرضها على الشاشة.. إنّ هذه العمليّة تسمّى "التحويلات المتتابعة" Transformation Pipeline.. وطبعا هناك رياضيّات مرعبة خلف هذه العمليّة، وإن كنّا سنستعين عليها بالعديد من الدوالّ الجاهزة، التي ستجعل الأمر بالنسبة لنا أبسط كثيرا.

**تحويل الإحداثيّات World Ttransform:**

افترض أنّ لدينا خمسة مكعّبات في أماكن مختلفة، ونريد عرض لقطة لها معا على الشاشة.. في البداية يمكننا تمثيل هذه المكعّبات بـ 40 رأسا (8 لكلّ مكعّب).. بعد هذا سنطلب من Direct3D أن يرسمها لنا.

إنّ هذا صحيح وسيفي بالغرض.. ولكنّ هناك مشكلة: ماذا لو كانت المكعّبات الخمسة متماثلة، وكلّ ما هناك هو أنّها في مواضع مختلفة؟.. في هذه الحالة نكون قد احتفظنا بخمسة أضعاف المعلومات المطلوبة، ففي حقيقة الأمر ليس لدينا سوى مكعّب واحد، تمّ رسمه في خمسة أماكن مختلفة بزوايا مختلفة!

إذن فالأفضل أن ننشئ نسخة واحدة من المكعّب، مركزها نقطة الأصل (0،0،0).. بعد ذلك يمكننا أن نقوم بالتحويلات اللازمة لتغيير موضع هذا المكعّب، مع إمكانيّة تكبيره وتصغيره وتدويره.. بهذه الطريقة يمكننا أن ننشئ أيّ عدد من النسخ من هذا المكعّب بمنتهى البساطة والسرعة.

**تحويلات الكاميرا Camera Transforms:**

كيف سيشعر مستخدم الكمبيوتر بأنّ ما يشاهده مجسّم، إذا لم يكن باستطاعته أن يتحرّك بحريّة بداخله؟

في هذه الحالة يجب أن تكون هناك وسيلة لتغيير زوايا المشهد (بخلاف أن يدخل المستخدم داخل الشاشة بنفسه بالطبع!!).. هذه الوسيلة هي الكاميرا، التي تتيح لك اختيار الموقع الذي تنظر منه للمشهد.

**تحويلات المنظور Projection Transforms:**

لن يكتمل تجسيم المشهد، ما لم يكن باستطاعتنا تحديد درجة تناسب الأحجام مع المسافات، ودرجة التركيز على المشهد، ودرجة تحدّب زاوية الرؤية... إلخ.

**العناصر الهندسيّة Meshes والنماذج Models:**

يعبّر مصطلح Mesh عن أيّ عنصر هندسيّ، مثل الرءوس Vertices و المثلّثات Triangles... إلخ..

ويمكن جمع هذه العناصر معا لتكوين مجسّم معقّد (مثل جسم الإنسان).. وكما ذكرنا من قبل، سيكون من العبث محاولة كتابة معادلات لرسم الأشكال المعقّدة، والأسهل أن يتمّ تصميمها في تطبيقات الرسوم المجسّمة، واستيرادها.

وهناك مصطلح آخر يجب أن تعرفه، ذلك هو مصطلح Model، الذي يعبّر عن عنصر هندسيّ وخصائصه المختلفة (اللون، الإضاءة، الخامة، ... إلخ).

**استخدام Direct3D9**

* [**مرحبا في VB:**](file:///G:\advanced\3-%20استخدام\1-%20مرحبا.htm)
* [**كيف تعدّ جهازك للتعامل مع DirectX9:**](file:///G:\advanced\3-%20استخدام\2-%20كيف%20تعدّ%20جهازك.htm)

**مرحبا في VB:**

قديما كانت الوسيلة المثلي لاستخدام Direct3D هي عن طريق استدعاء دوال API من لغة C++.. كان ذلك كذلك، إلى أصدرت ميكروسوفت DirecX7، وسمحت فيه لمبرمجي VB بالتعامل مع DirectX بطريقة مشابهة لتقنية COM.

والآن مع DirectX9 تطوّر الأمر كثيرا، حيث صار بإمكان مبرمج VB.Net أن يستخدم DirextX9 بطريقة مشابهة لاستخدام خلايا إطار العمل Framework Classes، وذلك عن طريق ما يسمّى بالكود المنظّم Managed-code، وهو ذلك الذي تديره بيئة VS.Net حتّى لا يكون مرتبطا بنوع نظام التشغيل أو مكوّنات الجهاز، وحتّى يكون آمنا في تعامله مع الذاكرة المؤقّتة RAM.

وللمقارنة، فإنّ استدعاء دوال API هو كود غير منظّم Unmanaged-code، وذلك لاعتماده على إصدار نظام التشغيل.. كذلك فإنّ حجز مساحة من الذاكرة مباشرةً هو كود غير منظّم، لأنّه قد يؤدّي إلى استهلاك مساحة الذاكرة لو لم يتمّ تحرير المساحات المحجوزة بعد الانتهاء من استخدامها.

ملحوظة:

قلنا مرارا إن ميكروسوفت أصدرت VS.NET وفي نيتها جعل التطبيقات المنشأة بها قابلة للعمل على أنظمة التشغيل المختلفة (مثل Linux).. للأسف، لن يكون ذلك سهلا بالنسبة للتطبيقات الخاصة بـ DirectX، لأنّه مصمّم خصّيصا للويندوز!

**كيف تعدّ جهازك للتعامل مع DirectX9:**

طبعا لا بدّ أن تتوافر لديك VS.Net!!.. بعد ذلك يجب أن تدخل موقع ميكروسوفت على الإنترنت:

[www.msdn.microsoft.com](http://www.msdn.microsoft.com/)

وتدخل قسم إنزال البرامج Downloads، ومنه تنزل DirectX9 SDK، وهي حوالي 230 ميجا، وهو حجم كبير بالنسبة لسرعات الإنترنت (استغرق منّي عدّة ساعات لإنزاله بتقنية DSL).. تصرّف.. المهمّ أن تحصل على هذا المكوّن.. ولا تنخدع وتنزل المكوّن الخاص بـ VB فقط، فعندما جرّبته لم يعمل!!.. ولكن من المحتمل طبعا أن تكون ميكروسوفت قد تداركت هذا الخطأ.. أنت بالخيار.

ملحوظة1:

يجب أن يكون إطار العمل Framework معدّا أولا على جهازك (يتمّ ذلك عند إعداد VS.Net)، قبل محاولة إعداد DirectX9 SDK، وإلا فلن يتمّ إعداد كلّ الملفّات المطلوبة على جهازك، ولن يعمل DirectX من خلال VB، حتّى لو أعددت إطار العمل بعد ذلك.. إنّ الترتيب مهمّ جدّا.

ملحوظة2:

لكي تعمل تطبيقاتك على جهاز العميل، يجب أن يقوم برنامج الإعداد بالخطوتين التاليتين بالترتيب:

1- إعداد إطار العمل على جهاز العميل.. إنّ ميكروسوفت تسمح بتداول نسخة مجانية من إطار العمل، يجب أن يتمّ تضمينها ببرنامج الإعداد.

2- إعداد DirectX9 Runtimes (ليس SDK كلّها) على جهاز العميل.

ملحوظة3:

بعد أن تعدّ DirectX9، تصفّح المجلّد الذي أنزلته به، وحاول أن تجرّب الأمثلة الموجودة به.. وستجد مجلّدا اسمه Bin به كلّ الملفات التنفيذيّة لهذه الأمثلة.. لو حدثت مشاكل في تشغيل هذه الأمثلة، فاعلم أنّ كارت الشاشة الخاص بك غير مناسب، وعليك تغييره أولا، والحصول على كارت يدعم الرسوم ثلاثيّة الأبعاد.. حينئذ لا أعرف إن كان عليك أن تعيد إعداد DirectX9 ليتناسب مع المكوّن الجديد أم لا، فأنا لم أجرّب هذه النقطة!

**مشروع تعليميّ**

* [**مكعبان دوّاران:**](file:///G:\advanced\4-%20مشروع%20تعليميّ\1-%20مكعبان%20دوّران.htm)
* [**إضافة مرجع لـ Direct3D:**](file:///G:\advanced\4-%20مشروع%20تعليميّ\02-%20إضافة%20مرجع.htm)
* [**تعريف المتغيرات:**](file:///G:\advanced\4-%20مشروع%20تعليميّ\03-%20تعريف%20المتغيرات.htm)
* [**وضع القيم الابتدائيّة لـ Direct3D9:**](file:///G:\advanced\4-%20مشروع%20تعليميّ\04-%20وضع%20القيم%20الابتدائيّة.htm)
* [**تكوين المجسّمات في الذاكرة:**](file:///G:\advanced\4-%20مشروع%20تعليميّ\05-%20تكوين%20المجسّمات.htm)
* [**الخامات Textures:**](file:///G:\advanced\4-%20مشروع%20تعليميّ\06-%20الخامات.htm)
* [**مصفوفات التحويل Transformation Matrices:**](file:///G:\advanced\4-%20مشروع%20تعليميّ\07-%20مصفوفات%20التحويل.htm)
* [**تحديث الإطار Frame Update:**](file:///G:\advanced\4-%20مشروع%20تعليميّ\08-%20تحديث%20الإطار.htm)
* [**استخدام الخليّة:**](file:///G:\advanced\4-%20مشروع%20تعليميّ\09-%20استخدام%20الخليّة.htm)
* [**ما الذي تعلّمناه في هذا المشروع:**](file:///G:\advanced\4-%20مشروع%20تعليميّ\10-%20ما%20الذي%20تعلّمناه.htm)

**مكعبان دوّران:**

أعرف أنّك قد مللت من كلّ هذا الكلام النظريّ الجافّ.. تعال إذن نبدأ رحلة المتعة.

وكبداية، سنحاول أن نتعلّم، عن طريق إنشاء مكعّبين يدوران على الشاشة، كما في الصورة التالية:



ابدأ مشروعا جديدا، وأسمه Basic\_D3D9.. أضف لهذا المشروع خليّة Class وأسمها CSampleGraphicsEngine.vb.

**إضافة مرجع لـ Direct3D:**

لتتمكن من استخدام Direct3D في مشروعك، يجب أن تضيفه كمرجع للمشروع.

اضغط القائمة الرئيسيّة Project، واختر الأمر Add Reference.. ستظهر لك نافذة تحتوي على كلّ المراجع التي يمكن إضافتها.. لو كنت قد أعددت DirectX9 SDK، فستجد العناصر التالية في القائمة:

**Microsoft.DirectX**

**Microsoft.DirectX.Direct3D**

**Microsoft.DirectX.Direct3DX**

اختر هذه العناصر، بحيث تظهر في القائمة السفليّة، ثمّ اضغط موافق.

بعد ذلك عليك باستيراد هذه المراجع في بداية كلّ ملفّ ستستخدمها فيه:

**Imports Microsoft.DirectX**

**Imports Microsoft.DirectX.Direct3D**

**Imports System.Math**

لاحظ أنّ السطر الثالث يستورد خليّة الرياضيّات، وهي تنتمي لإطار العمل، وذلك لأنّنا سنحتاج للدوال الرياضيّة في مشروعنا.

**تعريف المتغيرات:**

الخطوة التالية هي أن نقوم بتعريف متغيّرات عامّة للخليّة.. فلنبدأ بالكائنات الأساسيّة:

**Private D3DRoot As Manager**

**Private D3DDev As Device**

**Private D3DHelp As D3DX**

إنّ كائن الإدارة Manager يمثّل مكتبة Direct3D، وبالتالي فهو نقطة انطلاقنا التي يجب أن نبدأ منها عملنا.

إنّ هذا الكائن يتحكّم في الأجهزة Devices، عن طريق العديد من الوسائل التي تنشئ وتغلق الأجهزة وتتحكّم في عملها.

ولكن ما هو الجهاز Device؟

إنّه ببساطة، كائن يمثّل أحد مكوّنات الكمبيوتر الماديّة التي تستخدمها.. إنّ هذا معناه أنّنا نتحكّم في مكوّنات الجهاز مباشرة.

وأخيرا لدينا مكتبة المساعدة D3DX، وهي عبارة عن مجموعة من دوال الخدمات التي ترافق كلّ إصدار من إصدارات DirectX، لتمنح المبرمج العديد من التسهيلات.

بعد ذلك سنقوم بتعريف مصفوفات التحويل Transformation matrices:

**Private matCube1 As Matrix**

**Private matCube2 As Matrix**

**Private matView As Matrix**

**Private matProj As Matrix**

وهي تستخدم في عمليات تحويل الإحداثيات المختلفة.

يلي ذلك تعريف الخاماتTextures :

**Private texCube1 As Texture**

**Private texCube2 As Texture**

**Private texMenu As Texture**

ثمّ الأشكال الهندسيّةGeometry :

**Private vbCube As VertexBuffer**

**Private vbMenu As VertexBuffer**

ثمّ الخطوطFonts :

**Private fntOut As Font**

وهناك مجموعة أخرى من المتغيّرات، أفضّل أن نشرحها في حينها.

**وضع القيم الابتدائيّة لـ Direct3D9**

أوّل خطوة في أيّ مشروع تنشئه بـ Direct3D، هي أن تعدّ أجزاء الجهاز المختلفة لتناسب الوظيفة التي ستؤدّيها، وذلك بوضع قيم ابتدائيّة لها:

* [**الخطوط العامّة:**](file:///G:\advanced\04-%20وضع%20القيم%20الابتدائيّة\1-%20الخطوط%20العامّة.htm)
* [**طور العرض Display Mode والتنسيق Format:**](file:///G:\advanced\04-%20وضع%20القيم%20الابتدائيّة\2-%20طور%20العرض.htm)
* [**المخزن الخلفيّ BackBuffer:**](file:///G:\advanced\04-%20وضع%20القيم%20الابتدائيّة\3-%20المخزن%20الخلفيّ.htm)
* [**حدث إنشاء الخليّة New Constructor:**](file:///G:\advanced\04-%20وضع%20القيم%20الابتدائيّة\4-%20حدث%20إنشاء%20الخليّة.htm)
* [**مخزن البعد الثالث Depth Buffer:**](file:///G:\advanced\04-%20وضع%20القيم%20الابتدائيّة\5-%20مخزن%20البعد%20الثالث.htm)
* [**نوع الجهاز Device Type:**](file:///G:\advanced\04-%20وضع%20القيم%20الابتدائيّة\6-%20نوع%20الجهاز%20Device%20Type.htm)
* [**دالة وضع القيم الابتدائيّة:**](file:///G:\advanced\04-%20وضع%20القيم%20الابتدائيّة\7-%20دالة%20وضع%20القيم%20الابتدائيّة.htm)

**الخطوط العامّة:**

ما يجب أن تعرفه هنا، هو أنّ وضع القيم الابتدائيّة لـ Direct3D يتبّع خطواتٍ تكاد تكون ثابتة في كلّ مرّة:

1- تحديد طور العرض display mode والتنسيق Format.

2- ضبط مخزن البعد الثالث (العمق) Depth Buffer.

3- ضبط أيّ اختيارات إضافيّة.

4- إنشاء جهاز Device.

5- إعداد حالات الرسم Render states.

6- إعداد مصفوفات التحويلات المبدئيّة.

**طور العرض Display Mode والتنسيق Format:**

توجد طريقتان في Direct3D لتكوين الأجهزة: فإمّا أن تحتلّ إحدى النوافذ Windowed Mode أو تحتلّ الشاشة كاملة Fullscreen Mode.. أعتقد أنّك تدرك الفارق بين النوعين.

ليس هذا فقط هو ما يهمّنا عند إنشاء الجهاز.. يهمّنا كذلك المساحة التي سنستخدمها من الشاشة.. هنا يجب أن تراعي طور العرض Display Mode، الذي يمثّل مدى دقّة العرض Resolution على الشاشة.. إنّ هذا سيؤثّر كثيرا على الكود الذي سنكتبه.

ونظرا لأنك تصمّم مجسّماتك في طور عرض معيّن، في حين أنّ مستخدم الكمبيوتر يستطيع تغيير طور عرض جهازه كما يحلو له، فإنّ أمامك اختيارين:

* إمّا أن تطلب من المستخدم تغيير طور العرض قبل استخدام برنامجك.
* وإمّا أن تقوم أنت بتغيير طور العرض برمجيّا قبل بدء برنامجك، ثمّ إعادته لما كان عليه بعد انتهاء البرنامج.. في هذه الحالة ستقع في مشكلة، وهي أنّ أطوار العرض ترتبط بنوعيّة كارت الشاشة، وقد لا يسمح الكارت الموجود بطور العرض الذي تريده.. إذن فعليك أن تتأكّد من قدرات كارت الشاشة في البداية.

تعال نكتب أوّل دالة في مشروعنا، وهي الدالة "هل طور العرض مناسب" isDisplayModeOkay.. ولكن كيف تعمل هذه الدالة؟

إنّها تستقبل منك ثلاثة معاملات:

عرض الشاشة (المعامل iWidth**)**.. وارتفاعها (المعامل iHeight) وعمق الألوان Depth (المعامل iWidth**).**

وهي المقاييسالتي تريد أن تتحقّق من أن جهاز العميل يمكن أن يتعامل معها بلا مشاكل.

في البداية ستتحقّق الدالة من التنسيق Format الذي يناسب عمق الألوان.

إنّ Direct3D9 يقدّم العديد من التنسيقات الجديدة لمعلومات الألوان.. وعليك أن تقرّر إذا كنت تفضّل استخدام 16 خانة ثنائيّة 16 bits أم 32، لحفظ معلومات اللون لكلّ نقطة Pixel.. طبعا استخدام 16 خانة أوفر في المساحة وأسرع نوعا في التنفيذ، ولكنّ استخدام 32 خانة أكثر كفاءة على حساب مساحة الذاكرة (ضعف المساحة).. وكمثال: الشاشة التي مساحتها 1024×768 بتنسيق 32 خانة لكلّ نقطة تحتاج إلى مساحة ذاكرة تساوي 1024×768×32 = 3 ميجا بايت.

ما يهمّنا الآن هو نوعان من التنسيقات:

|  |  |
| --- | --- |
| R5G6B5 | في هذا التنسيق يتمّ تخزين مكوّنات لون كلّ نقطة في 16 خانة:  5 لتخزين نسبة لأحمر، و 6 للأخضر و 5 للأزرق. |
| X8R8G8B8 | في هذا التنسيق يتمّ تخزين مكوّنات لون كلّ نقطة في 32 خانة:  8 خانات غير مستخدمة، و 8 لتخزين نسبة لأحمر، و 8 للأخضر و 8 للأزرق. |
| A1R5G5B5 | في هذا التنسيق يتمّ تخزين مكوّنات لون كلّ نقطة في 16 خانة:  خانة واحدة لتخزين درجة الشفافية، و 5 لتخزين نسبة لأحمر، و 5 للأخضر و 5 للأزرق. |

إنّ هناك حوالي 40 تنسيقا مختلفا، ولكنّك تستطيع أن تفهمها كلّها بمجرّد النظر، فهي تحتوي على الحروف A (ويرمز لدرجة الشفافية)، و R (ويرمز للأحمر)، و G (ويرمز للأخضر) و B (ويرمز للأزرق)، و X (ويرمز للخانات غير المستخدمة).. ويلي كلّ حرف عدد الخانات التي يستخدمها.

هذا هو الكود الذي يحدّد التنسيق المناسب:

**Dim fmt As Format**

**' اختيار تنسيق مناسب لعمق اللون**

**Select Case iDepth**

**Case 16**

**fmt = Format.R5G6B5**

**Case 32**

**fmt = Format.X8R8G8B8**

**End Select**

بعد هذا تستخدم الدالة كائن الإدارة Manger للحصول على كلّ الموصلات Adapters التي تستخدم للتحكّم في الشاشة، وذلك باستخدام الخاصيّة Adapters، التي تعيد مجموعة Collection تحتوي على كائنات من النوع "معلومات الموصّل" AdapterInformation، يمثّل كلّ منها أحد الموصّلات المعدّة على النظام.

ثمّ تقوم الدالة بالمرور عبر كلّ كائنات معلومات الموصل واحدا تلو آخر:

**Dim AdapterInfo As AdapterInformation**

**Dim D3Dr As Manager**

**For Each AdapterInfo In D3Dr.Adapters**

**Next**

حيث سنحاول في داخل هذه الجملة التكراريّة، الحصول على أطوار العرض التي تسمح بالتنسيق المطلوب (عمق اللون المطلوب) في كلّ موصّل، وذلك باستخدام الوسيلة SupportedDisplayModes()، والتي تستقبل نوع التنسيق كمعامل، وتعيد مجموعة Collection تحتوي على كائنات من النوع "طور العرض" DisplayMode، يمثّل كلّ منها أحد أطوار العرض الملائمة للتنسق المطلوب.

**Dim DispMode As DisplayMode**

**For Each DispMode In AdapterInfo.SupportedDisplayModes(fmt)**

**Next**

آخر خطوة هي اختبار مساحة العرض التي تقدّمها لنا كلّ أطوار العرض المتاحة، وذلك باستخدام خاصيتي Width و Height لكائن طور العرض.

**If DispMode.Width = iWidth AndAlso DispMode.Height = iHeight Then**

**Return True**

**End If**

إنّ كلّ ما نريده هو أن نتأكّد أنّ هناك طور عرض واحدا على الأقل في أيّ موصّل، يمتلك السمات المطلوبة.. فإذا وجدنا واحدا، فلا داعي لإكمال التحقّق من باقي الأطوار في باقي الموصلات، وعلينا أن نغادر الدالة ونعيد القيمة True، دلالة على أنّ كلّ شيء على ما يرام.

ها هو ذا كود الدالة كاملا:

**Public Shared Function isDisplayModeOkay(iWidth As Integer, \_**

**iHeight As Integer, iDepth As Integer) As Boolean**

**Try**

**Dim AdapterInfo As AdapterInformation**

**Dim DispMode As DisplayMode**

**Dim fmt As Format**

**Dim D3Dr As Manager**

**' اختيار تنسيق مناسب لعمق اللون**

**Select Case iDepth**

**Case 16**

**fmt = Format.R5G6B5**

**Case 32**

**fmt = Format.X8R8G8B8**

**End Select**

**' التأكّد من وجود طور عرض مناسب لمساحة العرض وعمق الألوان**

**For Each AdapterInfo In D3Dr.Adapters**

**For Each DispMode In \_**

**AdapterInfo.SupportedDisplayModes(fmt)**

**If DispMode.Width = iWidth AndAlso \_**

**DispMode.Height = iHeight Then**

**Return True**

**End If**

**Next**

**Next**

**' لو وصل التنفيذ لهذا السطر، فهذا يعني عدم العثور على طور عرض مناسب**

**Throw New Exception("لم يتمّ العثور على دقّة عرض مناسبة")**

**Catch DXErr As DirectXException**

**'DirectX لتلافي حدوث أي خطإ غير متوقع في**

**Return False**

**Catch Err As Exception**

**'لتلافي حدوث أي خطإ آخر غير متوقّع**

**Return False**

**End Try**

**End Function**

**المخزن الخلفيّ BackBuffer:**

نحتاج هنا لإيضاح مفهوم المخزن الخلفيّ BackBuffer، الذي يظهر في أكثر من خاصيّة من خصائص سجلّ "المعاملات الحاليّة" PresentParameters Structure، والذي سنستخدمه في الكود بعد قليل.

أنت تعرف أنّ حركة المجسّم على الشاشة تتمّ عن طريق إعادة رسمه في مواضع متتالية، بسرعة لا تسمح للعين بملاحظة هذه العمليّة (أكثر من 20 مرة في الثانية الواحدة).. ولكي يتمّ ذلك بكفاءة، يجب أن يتمّ رسم المجسّم كاملا أولا في مخزن خفيّ عن عين المستخدم، ثمّ يتمّ إظهار اللقطة على الشاشة كاملة.. بخلاف هذا، سيتمّ رسم المجسّم نقطةً نقطةً أمام المستخدم، وهي عمليّة تتّسم بشيء من البطء، ممّا سيجعل المجسّم يبدو لعين المستخدم وكأنّه يرتعش.

إذن فعليك في كلّ مرّة تستخدم فيها Direct3D، تعريف مخزن خلفيّ BackBuffer بنفس سمات الشاشة التي سترسم عليها.

دعنا نرى.

**حدث إنشاء الخليّة New Constructor:**

سننشئ تعريفين لهذا الحدث.. أحدهما يستقبل معاملا واحدا، يمثّل النموذج الذي سيتمّ الرسم عليه، بينما الآخر يستقبل المزيد من المعاملات، لتمثّل عرض وارتفاع الشاشة وعمق الألوان.

ونظرا لأنّ الصيغة الأولى لا تمدّ الخليّة بأيّ معلومات حول طريقة العرض، فسنفترض أنّ مستدعيها يريد عرض الرسوم داخل نافذة Windowed-mode، بينما في الصيغة الثانية سنفترض أنّ العرض سيكون في كلّ الشاشة Fullscreen.

دعنا أولا نعرّف بعض المتغيّرات الخاصة على مستوى الخليّة، لنضع فيها بعض القيم التي يستقبلها حدث إنشاء الخليّة:

**Private rTarget As Form ' النموذج الذي سيتمّ الرسم عليه**

**Private bWindowed As Boolean ' طور الشاشة (كاملة/نافذة)**

**Private sDispInfo As String** **' سنكتب في هذا المتغيّر معلومات حول طريقة العرض**

**Private bInitOkay As Boolean = False** **' متغيّر يمثّل نجاح وضع القيم الابتدائيّة أم لا**

ها هي ذي الصيغة الأولى:

**Public Sub New(ByVal Target As System.Windows.Forms.Form)**

**Try**

**Dim d3dPP As New PresentParameters()**

**d3dPP.Windowed = True**

**bWindowed = True**

**d3dPP.SwapEffect = SwapEffect.Discard**

**d3dPP.BackBufferCount = 1**

**d3dPP.BackBufferFormat = \_**

**D3DRoot.Adapters(0).CurrentDisplayMode.Format**

**d3dPP.BackBufferWidth = Target.ClientSize.Width()**

**d3dPP.BackBufferHeight = Target.ClientSize.Height()**

**sDispInfo = "[WINDOWED] " + \_**

**Target.ClientSize.Width.ToString + "x" + \_**

**Target.ClientSize.Height.ToString + " " + \_**

**d3dPP.BackBufferFormat.ToString()**

**rTarget = Target**

**initialiseDevice(CType(Target, Control), d3dPP)**

**Catch err As Exception**

**bInitOkay = False**

**Throw New Exception("فشل إعداد محرك الرسوم.")**

**End Try**

**End Sub**

إنّ معظم ما فعلناه في هذا الإجراء، هو ملء سجلّ "المعاملات الحاليّة" PresentParameters بالمعلومات اللازمة، وسنرى فيم سنستخدمه، وذلك عند شرح الدالة initialiseDevice، التي بدورها تمّ استدعاؤها في هذا الإجراء.

والآن، تعال نرى الصيغة الأخرى لحدث الإنشاء، تلك الخاصة بالعرض على كل ّالشاشة:

**Public Sub New(ByVal Target As Form, ByVal iWidth As Integer, \_**

**ByVal iHeight As Integer, ByVal iDepth As Integer)**

**Try**

**Dim d3dPP As New PresentParameters()**

**If Not isDisplayModeOkay(iWidth, iHeight, iDepth) Then**

**Throw New Exception("عرض الشاشة كاملة غير ممكن")**

**End If**

**bWindowed = False**

**d3dPP.BackBufferWidth = iWidth**

**d3dPP.BackBufferHeight = iHeight**

**d3dPP.BackBufferCount = 1**

**d3dPP.SwapEffect = SwapEffect.Copy**

**d3dPP.PresentationInterval = PresentInterval.Immediate**

**Select Case iDepth**

**Case 16**

**d3dPP.BackBufferFormat = Format.R5G6B5**

**Case 32**

**d3dPP.BackBufferFormat = Format.X8R8G8B8**

**Case Else**

**Throw New Exception("عمق اللون غير مقبول")**

**End Select**

**rTarget = Target**

**initialiseDevice(CType(Target, Control), d3dPP)**

**Catch err As Exception**

**bInitOkay = False**

**Throw New Exception("Could not initialize graphics engine")**

**End Try**

**End Sub**

بخلاف عدد المعاملات، هناك اختلافان جوهريان بين الصيغتين:

الأوّل يتمثّل في عدم استخدام أبعاد النموذج، واستخدام المواصفات المرسلة كمعاملات بدلا من ذلك، طبعا بعد التأكّد من صلاحيتها للعرض، باستخدام الدالة isDisplayModeOkay.

والثاني يتمثّل في استخدام خاصيّة "مدى التمثيل" PresentationInterval الخاصّة بالسجلّ PresentParameters لإجبار Direct3D على عرض المجسّم مباشرة على الشاشة بدون أيّ تأخير، بمجرّد اكتمال رسمه.

**مخزن البعد الثالث Depth Buffer:**

إنّ ما يتمّ رسمه على الشاشة ما هو إلا صورة ثنائيّة البعد.. فإذا رسمت مجسّما على الشاشة، فإنّه قد يتقاطع مع مجسّم آخر مرسوم من قبل، ممّا يعمل على إخفاء جزء منه.. إنّ هذا يضع على عاتقك عبئا ثقيلا، وهو حتميّة رسم المجسّمات بترتيب بُعدها: الأبعد عن الكاميرا، فالأقرب، فالأقرب.... وذلك حتّى لا يحدث أيّ تأثير غير منطقيّ عندما يخفي أحد المجسّمات جزءا من مجسّم آخر.

هنا يأتي دور مخزن البعد الثالث Depth Buffer (ويسمّى أيضا Z-Buffer)، فهو يعمل على إضافة بعد ثالث تخيّليّ، بحيث يحتفظ فيه بمعلومات عن عمق كلّ مجسّم يتمّ رسمه.. وعند رسم أيّ مجسّم جديد، يقوم Direct3D بالتحقّق من البعد الثالث لكلّ نقطة.. فإذا كانت النقطة المرسومة سابقا أقرب للكاميرا من النقطة المراد رسمها، تظلّ النقطة القديمة كما هي، ولا يتمّ رسم النقطة الجديدة على الشاشة.. ولو كانت النقطة الجديدة أقرب للكاميرا من النقطة القديمة، يتمّ رسمها بدلا منها، بحيث يبدو أنّ المجسّم الجديد قد أخفى كلّ أو بعض المجسّم القديم.

لاحظ أنّ هذا يستلزم عددا كبيرا من عمليات المقارنة، لهذا فإنّ Direct3D يستخدم بعض الخوارزميّات لتوقّع الأوجه الخلفيّة للمجسّمات (التي لا تواجه الكاميرا) ويتجاهل رسمها، لأنّها في كلّ الأحوال لن تظهر أمام الرائي من هذه الزاوية.

معنى هذا أنّ استخدام مخزن البعد الثالث يسمح لنا برسم المجسّمات دون اعتبار ترتيبها، ففي كلّ الأحوال سيبدو المشهد النهائيّ تماما كما أردناه.. والآن كلّ ما علينا هو استخدام الجملة التالية لتفعيل هذه الإمكانيّة:

**D3DDev.RenderState.ZBufferEnable = True**

**نوع الجهاز Device Type:**

قلنا إنّ الكائن Manager هو نقطة انطلاقنا، حيث سنستخدمه لإنشاء الأجهزة التي سنتعامل معها.

شيء أخر نحتاج لإيضاحه، هو ما يختصّ بنوع الجهاز.. إنّ هناك أربعة أنواع، تحدّد الطريقة التي سيتمّ بها إجراء العمليّات على نقاط الصورة:

|  |  |
| --- | --- |
| Software | المشغّل الدقيق الخاصّ بالكمبيوتر هو المسئول عن إجراء عمليّات التحويل والإضاءة. |
| Hardware | كارت الشاشة هو المسئول عن إجراء عمليّات التحويل والإضاءة. |
| Mixed | مزيج من النوعين السابقين: المشغّل الدقيق وكارت الشاشة سيشتركان معا في معالجة الرسوم. |
| Pure | كارت الشاشة هو المسئول عن معالجة الرسوم لأقصى درجة ممكنة. |

ويمكن ترتيب الأنواع السابقة من حيث الأفضليّة (السرعة وكفاءة الأداء) كالتالي: Hardware ـ Pure ـ Mixed ـ Software.

ويجب أن تنتبه إلى أنّ بعض هذه الأنواع قد لا يكون متاحا على جهاز العميل.. لهذا يجب أن نستخدم الوسيلة "اقرأ قدرات الجهاز" GetDeviceCaps الخاصّة بكائن الإدارة Manger.. هذه الوسيلة تستقبل معاملان: رقم الموصّل Adapter، ونوع الجهاز.. وتعيد هذه الوسيلة كائنا من النوع Caps:

**D3DCaps = D3DRoot.GetDeviceCaps(0, DeviceType.Hardware**

بعد ذلك استخدمنا هذا الكائن لنتأكّد من أنّ كارت الشاشة يستطيع معالجة التحويلات والإضاءة، وذلك بالسطر التالي:

**If D3DCaps.DeviceCaps.SupportsHardwareTransformAndLight Then**

فإذا كان هذا الشرط متحقّقا، يمكن إسناد معالجة التحويلات والإضاءة لكارت الشاشة:

**DevCreate = CreateFlags.HardwareVertexProcessing Or \_**

**CreateFlags.MultiThreaded**

وإذا كان هذا الشرط غير متحقّق، فسنسند هذه المهمّة للمشغّل الدقيق:

**DevCreate = CreateFlags.SoftwareVertexProcessing Or \_**

**CreateFlags.MultiThreaded**

لاحظ كذلك أنّنا سنرسل المتغيّر DevCreate كمعامل لحدث إنشاء الجهاز، وبذلك تؤثّر القيم التي وضعناها به على عمل الجهاز:

**D3DDev = New Device(0, DeviceType.Hardware, Target, \_**

**DevCreate, win)**

**دالة وضع القيم الابتدائيّة:**

سنعرّف الآن المزيد من المتغيّرات الخاصة على مستوى الخليّة:

**Private iFontSize As Integer ' حجم الخط الذي سيتم رسم النصوص به**

**Private lastFrameUpdate As Int32**

**Private sDevInfo As String ' اسم كارت الشاشة**

نصل الآن لكود الدالة initialiseDevice.. لا تجعل طوله يفزعك، فقد تعرّفنا بالفعل على كلّ أجزائه.. ها هو ذا:

**Private Sub initialiseDevice(ByVal Target As Control, \_**

**ByVal win As PresentParameters)**

**Try**

**Dim D3DCaps As Caps**

**Dim DevCreate As Integer**

**'1. التحقق من مخزن البعد الثالث، وتوصيفه**

**If D3DRoot.CheckDepthStencilMatch(0, \_**

**DeviceType.Hardware, win.BackBufferFormat, \_**

**win.BackBufferFormat, DepthFormat.D16) Then**

**win.AutoDepthStencilFormat = DepthFormat.D16**

**win.EnableAutoDepthStencil = True**

**Else**

**Throw New Exception("خطأ")**

**End If**

**'2. اختبار قدرات الجهاز**

**D3DCaps = D3DRoot.GetDeviceCaps(0, \_**

**DeviceType.Hardware)**

**' الاستفادة من إمكانيات مكونات الجهاز في التحويلات والإضاءة**

**If D3DCaps.DeviceCaps.SupportsHardwareTransformAndLight Then**

**DevCreate = CreateFlags.HardwareVertexProcessing Or \_**

**CreateFlags.MultiThreaded**

**Else**

**DevCreate = CreateFlags.SoftwareVertexProcessing Or \_**

**CreateFlags.MultiThreaded**

**End If**

**'3. محاولة إنشاء واجهة الجهاز**

**D3DDev = New Device(0, DeviceType.Hardware, \_**

**Target, DevCreate, win)**

**If D3DDev Is Nothing Then Throw New Exception("…")**

**'4. ضبط حالة الرسم والمعاملات الأخرى**

**' استخدام مخزن للبعد الثالث (العمق)**

**D3DDev.RenderState.ZBufferEnable = True**

**' لن نستخدم الإضاءة في هذا المثال**

**D3DDev.RenderState.Lighting = False**

**' السماح بشفافية الخامات**

**D3DDev.RenderState.SourceBlend = Blend.SourceAlpha**

**D3DDev.RenderState.DestinationBlend = Blend.InvSourceAlpha**

**' إعداد خليط الخامات**

**D3DDev.SamplerState(0).MinFilter = TextureFilter.Linear**

**D3DDev.SamplerState(0).MagFilter = TextureFilter.Linear**

**'5.تحميل الخامات**

**loadTextures(win.BackBufferFormat)**

**'6. إنشاء الرسوم الضرورية**

**loadGeometry()**

**'7. إعداد المصفوفات**

**'view matrix: تصف خصائص الكاميرا**

**D3DDev.Transform.View = Matrix.LookAtLH(New \_**

**Vector3(0, 0, -15), New Vector3(0, 0, 0), \_**

**New Vector3(0, 1, 0))**

**'projection matrix: تصف خصائص عدسات الكاميرا**

**D3DDev.Transform.Projection = \_**

**Matrix.PerspectiveFovLH(Math.PI / 4, 4 / 3, 1, 100)**

**'world matrix: تصف إحداثيات الأشكال وما سيتمّ عليها من تحويلات**

**D3DDev.Transform.World = Matrix.Identity()**

**'8. sort out fonts for on-screen rendering**

**iFontSize = 11**

**fntOut = New Font(D3DDev, \_**

**New Drawing.Font("Arial", iFontSize, FontStyle.Bold))**

**'9. ضبط المتغيّرات العامة**

**lastFrameUpdate = Environment.TickCount()**

**' الاحتفاظ باسم كارت الشاشة في متغيّر**

**sDevInfo = D3DRoot.Adapters(0).Information.Description**

**bInitOkay = True**

**Catch err As Exception**

**bInitOkay = False**

**Throw New Exception("فشل إعداد محرك الرسوم")**

**End Try**

**End Sub**

**تكوين المجسّمات في الذاكرة**

* [**كيفية تخزين المجسّمات**](file:///G:\advanced\05-%20تكوين%20المجسّمات\1-%20كيفية%20تخزين%20المجسّمات.htm)
* [**كيفيّة رسم المجسّمات:**](file:///G:\advanced\05-%20تكوين%20المجسّمات\2-%20كيفيّة%20رسم%20المجسّمات.htm)
* [**ترتيب الرءوس في المثلث:**](file:///G:\advanced\05-%20تكوين%20المجسّمات\22-%20ترتيب%20الرءوس.htm)
* [**تحميل المجسمات:**](file:///G:\advanced\05-%20تكوين%20المجسّمات\3-%20تحميل%20المجسمات.htm)

**كيفية تخزين المجسّمات:**

يمكنك أن تحفظ رءوس المجسّمات vertices بإحدى طريقتين:

1- مخازن الرءوس Vertex Buffers:

وهي بمثابة مصفوفة يوفرّها لك Diurect3D لمعالجة الرسوم بكفاءة عالية، حيث يتحكّم في كيفيّة تخزين الرسوم (إمّا في ذاكرة الكمبيوتر أو ذاكرة كارت الشاشة).. وحتّى تحصل على أفضل أداء من هذه الطريقة، استخدم مخزن الرءوس Vertex Buffer لتخزين عدد من الرءوس ما بينَ 100 و 5000.. فإذا زاد العدد عن ذلك، فاقسمه على أكثر من مخزن.. وإذا قلّ العدد عن ذلك، فضع رءوس أكثر من مجسّم معا في نفس المخزن.

2- باستخدام المصفوفات التقليديّة:

ولا ننصحك بهذه الطريقة، إلا إذا كنت تحتاج لقراءة وتغيير بيانات الرسوم مرارا وتكرارا، وهو ما يتّسم بنوع من البطء في الطريقة الأولى.

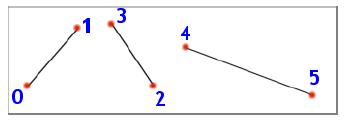
**كيفيّة رسم المجسّمات:**

حسنا.. لقد احتفظنا بالرءوس.. ولكن ماذا بعد؟

إنّ هناك 6 طرق للتعامل مع هذه الرءوس:

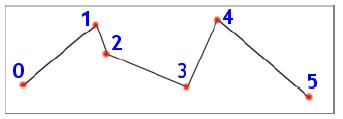
**1- قائمة الخطوط Line List:**

يتمّ رسم خط مستقيم بين كلّ زوج متتالٍ من الرءوس (دون أن تشترك رأس بين أكثر من خطّين).



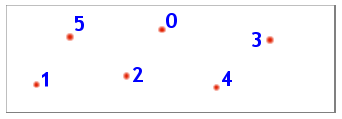
**2- شريط الخطوط Line Strip:**

يتمّ رسم خطّ مستقيم بين الرأس الحاليّة والرأس السابقة، بحيث يتمّ توصيل كلّ الرءوس معا بخطّ واحد متّصل، بدون المرور بأيّ نقطة أكثر من مرّة.



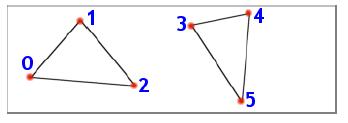
**3- قائمة النقاط Point List:**

يتمّ رسم كلّ الرءوس كنقاط مستقلّة.. ويمكنك استغلال هذه الإمكانيّة لرسم الأشكال الشبحيّة، كالدخان والماء والنار... إلخ.



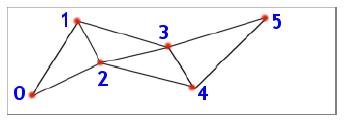
**4- قائمة المثلثات Triangle List:**

يتمّ توصيل كلّ ثلاث رءوس لتشكّل مثلّثا، دون أن تستخدم أيّ رأس في أكثر من مثلث.



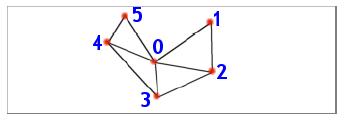
**5- شريط المثلثات Triangle Strip:**

يتم رسم مثلثات بين كلّ ثلاثة رءوس، مع اشتراك كلّ مثلثين متجاورين في أحد الأضلاع.



**6- مروحة المثلات Triangle Fan:**

يتمّ رسم كلّ المثلثات الممكنة، بحيث يكون الرأس الأوّل (رقم صفر) مشتركا بينها جميعا.



**تحميل المجسمات:**

سننشئ الآن الدالة loadGeometry، لتكون مسئولة عن تحميل الأشكال.. لا تجعل منظر الكود يفزعك، فما إن تقرأ شرحه، حتّى تجده في منتهى الوضوح:

**Private Sub loadGeometry()**

**Try**

**' تحميل الرسوم ثنائية البعد (قائمة الأوامر) في الرءوس**

**'[------------------------------------]**

**vbMenu = New VertexBuffer(GetType( \_**

**CustomVertex.TransformedTextured), 4, D3DDev, 0, \_**

**CustomVertex.TransformedTextured.Format, Pool.Default)**

**Dim v() As CustomVertex.TransformedTextured = CType( \_**

**vbMenu.Lock(0, 0), \_**

**CustomVertex.TransformedTextured())**

**Const iMBWidth As Integer = 158**

**Const iMBHeight As Integer = 93**

**v(0) = New CustomVertex.TransformedTextured(0, 86, 0, \_**

**1, 0, 2 / 127)**

**v(1) = New CustomVertex.TransformedTextured( \_**

**iMBWidth, 86, 0, 1, iMBWidth / 255, 2 / 127)**

**v(2) = New CustomVertex.TransformedTextured(0, \_**

**86 + iMBHeight, 0, 1, 0, iMBHeight / 127)**

**v(3) = New CustomVertex.TransformedTextured( \_**

**iMBWidth, 86 + iMBHeight, 0, 1, \_**

**iMBWidth / 255, iMBHeight / 127)**

**vbMenu.Unlock()**

**'[------------------------------------]**

**' تحميل الرسوم المجسمة في الرءوس**

**'[------------------------------------]**

**vbCube = New VertexBuffer( \_**

**GetType(CustomVertex.PositionTextured), \_**

**36, D3DDev, 0, \_**

**CustomVertex.PositionTextured.Format, \_**

**Pool.Managed)**

**Dim vCube As CustomVertex.PositionTextured() = \_**

**CType(vbCube.Lock(0, 0), \_**

**CustomVertex.PositionTextured())**

**'2a. إنشاء وجه المكعب العلويّ**

**vCube(0) = New CustomVertex.PositionTextured(-0.5, 0.5, \_**

**0.5, 0, 0)**

**vCube(1) = New CustomVertex.PositionTextured(0.5, 0.5, \_**

**0.5, 1, 0)**

**vCube(2) = New CustomVertex.PositionTextured(-0.5, 0.5, \_**

**-0.5, 0, 1)**

**vCube(3) = New CustomVertex.PositionTextured(0.5, 0.5, \_**

**-0.5, 1, 1)**

**vCube(4) = New CustomVertex.PositionTextured(-0.5, 0.5, \_**

**-0.5, 0, 1)**

**vCube(5) = New CustomVertex.PositionTextured(0.5, 0.5, \_**

**0.5, 1, 0)**

**'2b. إنشاء وجه المكعب السفليّ**

**vCube(6) = New CustomVertex.PositionTextured(-0.5, -0.5, \_**

**-0.5, 1, 0)**

**vCube(7) = New CustomVertex.PositionTextured(0.5, -0.5, \_**

**0.5, 0, 1)**

**vCube(8) = New CustomVertex.PositionTextured(-0.5, -0.5, \_**

**0.5, 0, 0)**

**vCube(9) = New CustomVertex.PositionTextured(0.5, -0.5, \_**

**0.5, 0, 1)**

**vCube(10) = New CustomVertex.PositionTextured(-0.5, \_**

**-0.5, -0.5, 1, 0)**

**vCube(11) = New CustomVertex.PositionTextured(0.5, -0.5, \_**

**-0.5, 1, 1)**

**'2c. إنشاء وجه المكعب الأيسر**

**vCube(12) = New CustomVertex.PositionTextured(-0.5, 0.5, \_**

**-0.5, 0, 0)**

**vCube(13) = New CustomVertex.PositionTextured(-0.5, \_**

**-0.5, -0.5, 1, 0)**

**vCube(14) = New CustomVertex.PositionTextured(-0.5, 0.5, \_**

**0.5, 0, 1)**

**vCube(15) = New CustomVertex.PositionTextured(-0.5, 0.5, \_**

**0.5, 0, 1)**

**vCube(16) = New CustomVertex.PositionTextured(-0.5, \_**

**-0.5, -0.5, 1, 0)**

**vCube(17) = New CustomVertex.PositionTextured(-0.5, \_**

**-0.5, 0.5, 1, 1)**

**'2d. إنشاء وجه المكعب الأيمن**

**vCube(18) = New CustomVertex.PositionTextured(0.5, 0.5, \_**

**-0.5, 0, 0)**

**vCube(19) = New CustomVertex.PositionTextured(0.5, 0.5, \_**

**0.5, 1, 0)**

**vCube(20) = New CustomVertex.PositionTextured(0.5, -0.5, \_**

**-0.5, 0, 1)**

**vCube(21) = New CustomVertex.PositionTextured(0.5, -0.5, \_**

**0.5, 1, 1)**

**vCube(22) = New CustomVertex.PositionTextured(0.5, -0.5, \_**

**-0.5, 0, 1)**

**vCube(23) = New CustomVertex.PositionTextured(0.5, 0.5, \_**

**0.5, 1, 0)**

**'2e. إنشاء وجه المكعب الخلفيّ**

**vCube(24) = New CustomVertex.PositionTextured(-0.5, 0.5, \_**

**-0.5, 0, 0)**

**vCube(25) = New CustomVertex.PositionTextured(0.5, 0.5, \_**

**-0.5, 1, 0)**

**vCube(26) = New CustomVertex.PositionTextured(-0.5, \_**

**-0.5, -0.5, 0, 1)**

**vCube(27) = New CustomVertex.PositionTextured(0.5, -0.5, \_**

**-0.5, 1, 1)**

**vCube(28) = New CustomVertex.PositionTextured(-0.5, \_**

**-0.5, -0.5, 0, 1)**

**vCube(29) = New CustomVertex.PositionTextured(0.5, 0.5, \_**

**-0.5, 1, 0)**

**'2f. إنشاء وجه المكعب الأماميّ**

**vCube(30) = New CustomVertex.PositionTextured(0.5, 0.5, \_**

**0.5, 1, 0)**

**vCube(31) = New CustomVertex.PositionTextured(-0.5, 0.5, \_**

**0.5, 0, 0)**

**vCube(32) = New CustomVertex.PositionTextured(-0.5, \_**

**-0.5, 0.5, 0, 1)**

**vCube(33) = New CustomVertex.PositionTextured(-0.5, \_**

**-0.5, 0.5, 0, 1)**

**vCube(34) = New CustomVertex.PositionTextured(0.5, -0.5, \_**

**0.5, 1, 1)**

**vCube(35) = New CustomVertex.PositionTextured(0.5, 0.5, \_**

**0.5, 1, 0)**

**vbCube.Unlock()**

**Catch err As Exception**

**MsgBox("loadGeometry(): " + Chr(13) + Chr(13) + err.ToString())**

**Throw New Exception("could not complete loadGeometry()")**

**End Try**

**End Sub**

تنقسم الدالة السابقة لقسمين: الأوّل يقوم بإنشاء قائمة بسيطة ثنائيّة البعد (مجرّد مستطيل سنعرض فيه بعض المعلومات عن وظائف بعض الأزرار)، والثاني يقوم بإنشاء المكعّب.

في كلتا الحالتين سنبدأ بحجز مخزن الرءوس VertexBuffer:

vbCube = New VertexBuffer(GetType(CustomVertex.PositionTextured), \_

36, D3DDev, 0, CustomVertex.PositionTextured.Format, Pool.Managed)

حيث المعامل الأوّل يمثّل نوع الرءوس، والثاني عددها، والثالث الجهاز الذي سيرسمها، والخامس نوع التنسيق (وسنستخدم هنا إحداثيات الموضع والخامات)، والأخير يعطى السلطة لـ Direct3D لإدارة الذاكرة المؤقّتة، سواء الخاصة بالكمبيوتر أو بكارت الشاشة.

بعد هذا سنغلق مخزن الرءوس أمام استخدامه من قبل أيّ تطبيق آخر، وسنحصل على مؤشّر Pointer يشير إلى بياناته، وذلك باستخدام الوسيلة vbCube.Lock(0, 0)، التي تعيد مصفوفة تقليديّة.. ونظرا لأنّنا نتعامل مع Vertex Buffer، فسنحوّل القيمة العائدة إلى مخزن رءوس من النوع الذي نستخدمه CustomVertex.PositionTextured:

Dim vCube() As CustomVertex.PositionTextured = \_

CType(vbCube.Lock(0, 0), CustomVertex.PositionTextured())

الآن يمكننا التعامل مع الخانات من 0 إلى 35 (36 خانة كما عرّفناها) في المصفوفة (المخزن) vCube.

الآن علينا أن نملأ هذه المصفوفة بالإحداثيّات.. طبعا عليك أن تحضر ورقة وقلما، وتصمّم المكعّب كما تريد، وتحسب إحداثيات رءوسه.

لاحظ أنّنا في هذا المثال استخدمنا طريقة قائمة المثلثات Triangle-List للتوصيل بين رءوس المكعّب.

آخر خطوة في هذه العمليّة، هي تحرير المخزن الذي أغلقناه أمام التطبيقات الأخرى.. إنّ هذه الخطوة ضروريّة، لأنّك Direct3D لو حاول أن يرسم المكعّب على الشاشة في حين أنّ مخزن الرءوس مغلق، فسيقوم بإطلاق استثناء ولن ينفّذ العمليّة:

**vbCube.Unlock()**

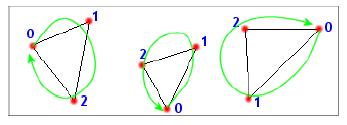
**ترتيب الرءوس في المثلث:**

لكي تحصل على الشكل الذي تريده، يجب عليك مراعاة الترتيب الذي تخزّن به الرءوس.. إنّ هذا الترتيب سيؤثّر على تحديد الأوجه الخلفيّة للمجسّم (تلك التي لا تظهر للرائي من الزاوية الحاليّة).. لقد ذكرنا من قبل أن Direct3D يتخلّص من الأوجه الخلفيّة، وهي عمليّة تعرف باسم "قتل الأوجه الخلفيّة" Back Face Culling، وذلك ليوفّر على نفسه وقت رسمها.

وبإمكانك استخدام الخاصيّة Device.RenderState.CullMode ـ كما سنرى لاحقا ـ لتخبر DirecteD بكيفيّة تحديد الأوجه الخلفيّة، وذلك عن طريق واحدة من قيم المرقّم Cull التالية:

|  |  |
| --- | --- |
| CounterClockwise | القيمة الافتراضيّة.. سيتمّ إزالة الأوجه الخلفيّة الموجودة في عكس اتجاه عقارب الساعة. |
| clockwise | سيتمّ إزالة الأوجه الخلفيّة الموجودة في اتجاه عقارب الساعة. |
| No culling | لن يتمّ محو الأوجه الخلفيّة.. إنّ هذا مفيد عند التعامل مع الأسطح شبه الشفافة، حيث يتمّ رؤية الخلفيات في هذه الحالة. |

وفي الصورة التالية، إذا استخدمنا CounterClockwise، فلن يتمّ رسم المثلّث الأوسط، لأنّ ترتيب رسم نقاطه في عكس اتجاه عقارب الساعة.



وعليك بالاحتراس عند تخزين رءوس المجسّمات، فعلى حسب الترتيب الذي ستضعها به في الذاكرة سيتمّ رسم أو محو المثلثات.

أعرف أنّك ما زلت متعجبا من احتياج Direc3D لهذه التقنية!!

حسنا.. إنّ قتل الأوجه الخلفيّة Culling يوفّر 40% من الوقت اللازم لرسم المجسّم بدون استخدام هذه التقنية.. فإذا لاحظت أنّ وجها واحدا فقط (من وجوه المكعّب الستّة) هو الذي يسهل تحديد كونه وجها خلفيّا، لأنّه يقع في المستوى (س ـ ص)، فلا بدّ أنّك ستدرك السرّ وراء احتياج Direct3D لتحديد اتجاه الإزالة.. إنّ عليه أن يزيل 3 أوجه من خمسة وجوه، ويحتاج منك لاختيار وسيلة سريعة لتحديدها.

والآن تعال نواصل مشروعنا.

**الخامات Textures**

الآن نتحدّث بمزيد من العمق عن الخامات:

* [**تنسيقات الخامات:**](file:///G:\advanced\06-%20الخامات\1-%20تنسيقات%20الخامات.htm)
* [**إحداثيّات الخامات Texture Coordinate:**](file:///G:\advanced\06-%20الخامات\2-%20إحداثيّات%20الخامات.htm)
* [**تحميل الخامات:**](file:///G:\advanced\06-%20الخامات\3-%20تحميل%20الخامات.htm)

**تنسيقات الخامات:**

ذكرنا من قبل أنّ الخامة هي صورة نقطيّة Bitmap يتمّ لصقها على جزء من المجسّم.

ولكي تحصل على أفضل أداء للخامة، اجعل مقاييسها ثنائيّة (أعداد مرفوعة للأس 2)، مثل 128×128، 256×256، 512×1024.... إلخ.. إنّ هذا سيسهّل على معظم أنواع كروت الشاشة تسريع التعامل مع هذه الخامات.

عامة ليس عليك أن تحفظ الخامات بهذه الأبعاد بنفسك، فـ Direct3D سيقوم بتحويلها تلقائيّا (ما لم تطلب أنت منه العكس).

لكن يجب عليك أن تتأكّد أولا من المدى الذي يستطيع فيه كارت الشاشة التعامل مع الخامات، حيث إنّ معظم الكروت تتعامل مع خامات مساحتها أصغر من أو تساوي 1024×1024، وهو كاف بالفعل في معظم الحالات.

ويتمّ حفظ الخامة على الكمبيوتر كصورة بالامتدادات المعروفة (BMP, TGA, GIF, JPG... إلخ).

وهناك امتداد آخر هو DDS (سطح الرسم المباشر Direct Draw Surface)، يتمّ فيه حفظ الصورة بنفس التنسيق الذي يتمّ تمثيلها به في الذاكرة بواسطة كارت الشاشة.. إنّ هذا يعنى تحميل الصورة من الملفّ بسرعة فائقة، كما يمنح مصمّم الصورة القدرة على تحديد عدد الخانات التي تمثّل درجة الشفافية كما يريد، كأن يجعلها خانتين 2 Bits بدلا من 8، دون أن يضيّع Direct3D وقتا طويلا في التحويلات.

فإذا كنت ترغب في إنشاء ملفّات بامتداد DDS، فيمكنك استخدام الأداة DXTex الموجودة مع DirectX9 SDK (اضغط قائمة البداية Start، برامج Programs، Microsoft DirectX 9.0 SDK، DirectX Utilities، DirectX Texture Tool).

وعند تحميل الخامة في ذاكرة كارت الشاشة، يتمّ حفظها بتنسيق معيّن.. ويمنحنا Direct3D 40 تنسيقا مختلفا، أسماؤها على شاكلة X8R8G8B8، وقد تعرّفنا من قبل على ما تعنيه الحروف R و G و B، وما تعنيه الأرقام التالية لكلّ منها.

ما يجب أن تلاحظه هنا، هو أنّ هناك مساحة مخصّصة من كارت الشاشة لتحميل الخامات (لو كان كارت الشاشة 32 ميجا، فستتوفر مساحة من 20 إلى 25 ميجا للخامات)، وعليك ألا تتجاوز هذه المساحة، وذلك بتحميل الخامات الضروريّة في الوقت المناسب، والتخلّص من الخامات التي انتهى دورها.

**إحداثيّات الخامات Texture Coordinate:**

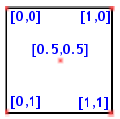
لكلّ رأس Vertex موقع ولون وإضاءة وإحداثيّات خامة.

إنّ الخامة ثنائيّة البعد 2D كما اتفقنا.. لهذا سيكون لكلّ رأس إحداثيان ثنائيا البعد، سنرمز لهما بالرمزين U و V.

إنّ هذا هو السبب الذي جعلنا نرسل خمس إحداثيّات أثناء ملء مصفوفة الرءوس في دالة تحميل الرسوم: 3 إحداثيّات الموضع، وإحداثيّان الخامة.

مع ملاحظ أنّ وحدة قياس الخامة تختلف عن وحدة قياس الموضع (يقاس بالنقطة Pixel)، حيث تقاس الخامات بأرقام عشريّة تنحصر بين 0.0 و 1.0، وأيّ أعداد خارج هذا النطاق لا يتمّ تمثيلها.

والصورة والجدول التاليين يوضّحان لك إحداثيّات بعض المواضع في الخامة:



|  |  |  |
| --- | --- | --- |
| U | V | الموضع |
| 0.0 | 0.0 | أعلى اليسار |
| 1.0 | 0.0 | أعلى اليمين |
| 0.0 | 1.0 | أسفل اليسار |
| 1.0 | 1.0 | أسفل اليمين |
| 0.5 | 0.5 | منتصف الوسط. |

إنّ لعدم استخدام الوحدات التقليديّة حكمة.. افترض أنّ المثلّث الذي سيتمّ لصق الخامة عليه يتحرّك بعيدا عن الكاميرا.. معنى هذا أن حجمه الظاهريّ سيقلّ.. في هذه الحالة لا جدوى من تحميل خامة ذات كفاءة عالية على مثلّث ضئيل المساحة.. لهذا فإنّ Drect3D يُنتج نسخا مختلفة من الخامة بدقّة مختلفة (256×256، 128×128، 64×64.... 2×2، 1×1)، ويختار مستوى الدقّة تبعا لحجم المثلّث وبعده عن الكاميرا.. لهذا فإنّ التعامل بإحداثيّات نسبيّة للخامة أفضل من التعامل مع إحداثيّات الشاشة، حيث سيقوم Direct3D بإجراء التحويلات المناسبة.

كما أنّ هذا يسمح لك أيضا بتغيير مساحة الخامة دون أن تغيّر الكود.. افترض أنّ لديك خامة 1024×1024 ولم يستطع كارت الشاشة التعامل معها.. في هذه الحالة يمكنك تحميلها بأبعاد أصغر، دون أن تخشى من حدوث مشكلة في تنفيذ الكود، فالإحداثيّات التي تستخدمها نسبيّة وليست حقيقيّة.

شيء آخر يجب أن تعرفه، هو أنّه من الممكن أن يكون للمثلّث الواحد أكثر من خامة (قد يصل الأمر إلى 8 خامات للمثلّث)، وطبعا يمكن ألا تستخدم الخامات أساسا.

**تحميل الخامات:**

نصل الآن لكود الدالة loadTextures، وهو بسيط للغاية.

لاحظ أنّنا سنستخدم ثلاث خامات، اثنتين للمكعّبين الذين سنرسمهما، وواحدة للقائمة.

ها هو ذا الكود:

**Private Sub loadTextures(ByVal AdapterFmt As Format)**

**' تحميل خامة القائمة**

**' يجب التأكّد أولا من صلاحية التنسيق للتعامل مع كارت الشاشة**

**If D3DRoot.CheckDeviceFormat(0, DeviceType.Hardware, \_**

**AdapterFmt, 0, ResourceType.Textures, \_**

**Format.A8R8G8B8) Then**

**texMenu = TextureLoader.FromFile(D3DDev, \_**

**Application.StartupPath + "\menubar.bmp", 256, \_**

**128, 1, 0, Format.A8R8G8B8, \_**

**Pool.Managed, Filter.None, \_**

**Filter.None, Drawing.Color.Magenta.ToArgb())**

**ElseIf D3DRoot.CheckDeviceFormat(0, \_**

**DeviceType.Hardware, AdapterFmt, 0, \_**

**ResourceType.Textures, Format.A1R5G5B5) Then**

**texMenu = TextureLoader.FromFile(D3DDev, \_**

**Application.StartupPath + "\menubar.bmp", \_**

**256, 128, 1, 0, Format.A1R5G5B5, Pool.Managed, \_**

**Filter.None, Filter.None, \_**

**Drawing.Color.Magenta.ToArgb())**

**Else**

**Throw New Exception("خطأ")**

**End If**

**' تحميل خامة المكعّبين**

**If D3DRoot.CheckDeviceFormat(0, DeviceType.Hardware, \_**

**AdapterFmt, 0, ResourceType.Textures, \_**

**Format.X8R8G8B8) Then**

**texCube1 = TextureLoader.FromFile(D3DDev, \_**

**Application.StartupPath + "\cube1.bmp", \_**

**256, 256, 1, 0, Format.X8R8G8B8, Pool.Managed, \_**

**Filter.Linear, Filter.Linear, 0)**

**texCube2 = TextureLoader.FromFile(D3DDev, \_**

**Application.StartupPath + "\cube2.bmp", \_**

**256, 256, 1, 0, Format.X8R8G8B8, Pool.Managed, \_**

**Filter.Linear, Filter.Linear, 0)**

**ElseIf D3DRoot.CheckDeviceFormat(0, DeviceType.Hardware, \_**

**AdapterFmt, 0, ResourceType.Textures, \_**

**Format.R5G6B5) Then**

**texCube1 = TextureLoader.FromFile(D3DDev, \_**

**Application.StartupPath + "\cube1.bmp", \_**

**256, 256, 1, 0, Format.R5G6B5, Pool.Managed, \_**

**Filter.Linear, Filter.Linear, 0)**

**texCube2 = TextureLoader.FromFile(D3DDev, \_**

**Application.StartupPath + "\cube2.bmp", \_**

**256, 256, 1, 0, Format.R5G6B5, Pool.Managed, \_**

**Filter.Linear, Filter.Linear, 0)**

**Else**

**Throw New Exception("…")**

**End If**

**End Sub**

أعتقد أنّ الكود واضح.. ربما ما يحتاج لقليل من الإيضاح هو الوسيلة CheckDeviceFormat، التي نتأكّد بها إذا ما كان التنسيق صالحا للاستخدام مع كارت الشاشة أم لا.. وتأخذ هذه الدالة عدّة معاملات، هي بالترتيب:

رقم الموصّل Adaper: وهو هنا أوّل موصّل (رقم صفر).

نوع الجهاز: ونحن نستخدم في هذا المشروع النوع Hardware.

تنسيق الموصّل: ونحن نستقبله في الدالة loadTextures كمعامل.. تذكّر أنّنا وضعناه في إحدى خصائص السجلّ PresentParameters من قبل.

خيارات الاستخدام: سنستخدم القيمة صفر، حيث لا يوجد أيّ استخدام خاصّ.

نوع المصدر: وهو هنا ResourceType.Textures.

نوع التنسيق: وهو التنسيق الذي نريد التأكّد من صلاحيته للاستخدام.. وسنجرّب أولا استخدام التنسيق X8R8G8B8، فإذا لم يكن متاحا فسنجرّب التنسيق R5G6B5.

لاحظ كذلك أن آخر معامل من معاملات الوسيلة TextureLoader.FromFile (والتي يتضح من اسمها أنّها تقوم بتحميل الخامة من ملف) يستقبل أحد الألوان.. إنّ هذا المعامل يقوم بنفس دور خاصيّة مفتاح الشفافية TransparencyKey في النموذج، حيث لا يتمّ رسم النقاط التي تحمل لون مفتاح الشفافية، وذلك لجعل منطقة من الرسم شفافة.

**مصفوفات التحويل Transformation Matrices:**

سنرى الآن كيف نستخدم مصفوفات التحويل لعمل نسختين مختلفتين من المكعّب الذي أنشأناه.. ولكن يجب عليك أولا أن تتذكّر هذه القاعدة الرياضيّة الهامّة للتعامل مع المصفوفات، تلك أنّ الترتيب مهمّ عند ضرب المصفوفات، بمعنى أنّ A×B لا يساوي B×A.. تذكّر هذا جيّدا عند قيامك بعمليّات تحويل متتابعة على أحد المجسّمات، فاختلاف الترتيب قد يؤدّي لاختلاف النتيجة.

عامّة هذه هي أهمّ العمليّات التي يمكن إجراؤها على المجسّمات:

1- تغيير الحجم Scale.

2- التدوير Rotation حول أيّ محور (س، ص، ع).

3- تغيير الموضع Translation.

ولكن... هل علينا أن نقوم بإجراء كلّ العمليّات الرياضيّة على المصفوفات بأنفسنا؟

لحسن الحظّ لا.. إنّ لدينا مكتبة رياضيّات Math library يمدّنا بها Direct3DX، وهي ستقوم بمعظم العمل المعقد بنفسها.. إنّ سجلّ المصفوفة Matrix structure يمدّنا بأكثر من 25 دالة تسهّل علينا التعامل مع مصفوفات التحويل.. والجدول التالي يوضّح لك سبعة من أهمّ هذه الدوال:

|  |  |
| --- | --- |
| Multiply | لضرب مصفوفتين معا (لتكوين مصفوفة جديدة تدمج عملتي التحويل معا). |
| Identity | مصفوفة الوحدة (كلّ عناصرها أصفار ما عدا قطرها، فكلّه آحاد).. عند ضرب هذه المصفوفة في أيّ مصفوفة أخرى لا يحدث أيّ تأثير (مثلما تضرب الواحد في أيّ عدد). |
| RotateX | لتدوير المجسّم حول محور السينات. |
| RotateY | لتدوير المجسّم حول محور الصادات. |
| RotateZ | لتدوير المجسّم حول محور العينات. |
| Scaling | تغيّر حجم المجسّم بنسبة معيّنة (ترسلها كمعامل). |
| Translation | لتغيير موضع المجسّم. |

انظر كيف نغيّر حجم المكعّب بالنسبة لجميع المحاور:

**matCube1 = Matrix.Multiply(matCube1, Matrix.Scaling(3, 3, 3))**

حيث matCube1 هي مصفوفة التحويل التي عرّفناها على مستوى الخليّة من قبل:

**Private matCube1 As Matrix**

لاحظ أنّ خطوة تغيير الحجم يمكن إجراؤها على أكثر من سطر كالتالي:

**Private M As Matrix**

**M = Matrix.Scaling(3, 3, 3)**

**matCube1 = Matrix.Multiply(matCube1, M)**

ما يجب أن تعرفه هنا، هو أنّ أيّ تغيير في المصفوفة يؤثّر على كلّ المجسّمات التي سيتمّ رسمها.

إذن كيف يمكن إحداث تأثيرات مختلفة على المجسّمات؟

بسيطة: كوّن مصفوفة التحويل الأولى وارسم المجسّم الأوّل.. هذا يعنى أنّه سيتأثّر بهذه التحويلات.. بعد ذلك كوّن مصفوفة التحويل الثانية وارسم المجسّم الثاني.. سيتأثّر المجسّم الثاني فقط بالتحويلات الجديدة، بينما المجسّم الأوّل ثابت كما هو.. ويمكن أن ترسم أكثر من مجسّم معا بنفس مصفوفة التحويل.. بعد أن تنتهي ستحصل على اللقطة التي تريدها، وبها كلّ مجسّم في الوضع المطلوب.

والآن تعال نكمل مشروعنا، لنرى كيف سنرسم المكعّبين ونحرّكهما.

**تحديث الإطار Frame Update**

قلنا إنّ الحركة تنتج عن عرض العديد من الصور المتتابعة بسرعة مناسبة.. إنّ كلّ صورة من هذه الصور تسمّى إطار Frame.. تعال الآن نرى كيف نكوّن هذه الإطارات، بحيث يبدو لمن يراها وكأنّ المكعبين يدوران:

* [**رسم الإطار:**](file:///G:\advanced\08-%20تحديث%20الإطار\1-%20رسم%20الإطار.htm)
* [**اعتبارات يجب مراعاتها عند الرسم:**](file:///G:\advanced\08-%20تحديث%20الإطار\2-%20اعتبارات%20يجب%20مراعاتها.htm)
* [**رسم المكعّبين على الشاشة:**](file:///G:\advanced\08-%20تحديث%20الإطار\3-%20رسم%20المكعّبين.htm)

**رسم الإطار:**

تعال نرى كود الإجراء oneFrameUpdate، وهو مسئول عن تكوين الإطار التالي للإطار المعروض حاليّا على الشاشة.. ولكننا سنعرّف بعض المتغيّرات على مستوى الخليّة أولا:

**Private cube1Angle As Single**

**Private cube2Angle As Single**

**Private Const cube1Speed As Single = 50.0F**

**Private Const cube2Speed As Single = 75.0F**

**Private Const cube1Size As Single = 2.0F**

**Private Const cube2Size As Single = 4.0F**

**Private iLastFPSCheck As Int32**

**Private Const iFPSProfileSpeed As Integer = 200 'ms**

**Private iFrameRate As Integer**

**Private iCurrCnt As Integer**

والآن، ها هو ذا كود الإجراء:

**Public Sub oneFrameUpdate()**

**If Not bInitOkay Then Throw New Exception("…")**

**Dim matTmp As Matrix**

**'حساب زوايا الدوران للمكعبين**

**cube1Angle += ((Environment.TickCount() - lastFrameUpdate) \_**

**/ 1000) \* cube1Speed**

**cube2Angle += ((Environment.TickCount() - lastFrameUpdate) \_**

**/ 1000) \* cube2Speed**

**lastFrameUpdate = Environment.TickCount()**

**'مصفوفة التحويل للمكعّب الأوّل**

**matCube1 = Matrix.Identity() ' مصفوفة الوحدة.. لا تأثير**

**' تغيير الحجم**

**matCube1 = Matrix.Multiply(matCube1, Matrix.Scaling(3, 3, 3))**

**' تدوير حول المحور السيني**

**matCube1 = Matrix.Multiply(matCube1, \_**

**Matrix.RotationX(cube1Angle \* (Math.PI / 180)))**

**' تدوير حول المحور الصادي**

**matCube1 = Matrix.Multiply(matCube1, \_**

**Matrix.RotationY(cube1Angle \* (Math.PI / 180)))**

**'مصفوفة التحويل للمكعّب الثاني**

**matCube2 = Matrix.Identity()**

**' تغيير الحجم**

**matCube2 = Matrix.Multiply(matCube2, Matrix.Scaling(4, 5, 0.75))**

**' تغيير موضع المكعّب**

**matCube2 = Matrix.Multiply(matCube2, \_**

**Matrix.Translation(-8, -4, 0))**

**' تدوير المكعّب حول محور العينات**

**matCube2 = Matrix.Multiply(matCube2, \_**

**Matrix.RotationZ(cube2Angle \* (Math.PI / 180)))**

**' حساب الإطار الحاليّ**

**If (Environment.TickCount() - iLastFPSCheck >= \_**

**iFPSProfileSpeed) Then**

**iLastFPSCheck = Environment.TickCount()**

**iFrameRate = iCurrCnt \* (1000 / iFPSProfileSpeed)**

**iCurrCnt = 0**

**End If**

**iCurrCnt += 1**

**End Sub**

والفكرة في هذا الكود، هي تغيير زاوية دوران المكعب بمقدار ثابت في كلّ ثانية (50 درجة في الثانية مثلا).. إنّ هذه الطريقة تضمن أداء ثابتا، فمهما زاد أو قلّ عدد الإطارات التي يتمّ رسمها في الثانية، فإنّ دورانا مقداره 50 درجة فقط هو الذي يحدث للمجسّم.. أمّا لو كنّا نعتمد على استخدام سرعة معيّنة للدوران (عدد الإطارات في الثانية)، وعلى أساس ذلك يتحدّد مقدار التغيير في زاوية الدوران، فإنّ هذا قد يؤدّي لعدم استقرار الحركة، فقد تكون سريعة جدّا على الكمبيوترات السريعة، وقد تكون بطيئة جدّا ومتقطّعة إذا كان هناك ما يعطّل الكمبيوتر أو كانت سرعته بطيئة.

ولحساب الوقت استخدمنا خليّة البيئة Environment Class، وهي تابعة لفضاء الاسم System في إطار العمل، ومهمّتها إمدادنا بمعلومات عن نظام التشغيل.

ونحن هنا نستخدم الخاصيّة "عدد الأجزاء من الألف من الثانية" Environment.TickCount، وهي تحسب الوقت المنقضي منذ تشغيل الويندوز إلى اللحظة الحاليّة، بدقّة أجزاء من الألف من الثانية.

لاحظ أنّنا نحتفظ بآخر وقت حدّثنا فيه الإطار في المتغيّر lastFrameUpdate، وبطرحه من الوقت الحالي يمكن معرفة الوقت المنقضي.. وطبعا نقسم على 1000 حتّى نقرّب الناتج لأقرب ثانية.

انظر إلى كيفيّة حساب زاوية الدوران للمكعّب الأوّل في الإطار الحاليّ:

**cube1Angle += ((Environment.TickCount() - lastFrameUpdate) / 1000) \_**

**\* cube1Speed**

المعادلة تقول ببساطة، إنّ الزاوية الجديدة تزيد عن الزاوية القديمة (لاحظ علامة +=)، بمقدار الوقت الذي انقضى (بالثانية) منذ رسم الإطار السابق، مضروبا في سرعة الدوران (معبّرا عنها بثابت، ليسهل عليك تغييرها في أيّ لحظة.. هذتا الثابت هنا يساوي 50).

إنّ هذا معناه أنّ دوران المجسّم سيكون مستقّرا وبسرعة ثابتة، فلو حدث شيء عطّل الكمبيوتر لثلاث ثوان مثلا، فستزيد زاوية الدوران بمقدار 3×50 = 150 مرّة واحدة، بحيث يبدو وكأنّ المجسّم كان يواصل حركته بطريقة طبيعيّة.. طبعا ليس من الشائع أن ينشغل الكمبيوتر عن تنفيذ برنامجنا لثلاث ثوان دفعة واحدة، والأحرى أن نتوقّع أن يتعطّل لأجزاء من مئات الأجزاء من الثانية على أسوأ تقدير!

وبعد أن نحسب زاويتي دوران المكعّبين، يجب أن نخزّن الوقت الحاليّ، لأنّ هذه هي اللحظة التي حدّثنا فيها الإطار الحاليّ، والتي ستدخل في حساب الإطار القادم:

**lastFrameUpdate = Environment.TickCount()**

الآن علينا أن نجرى التحويلات على المكعّبين.. تعال نرى ما سيحدث للمكعّب الأوّل:

1- تحجيم هذا المكعّب ليصبح 3×3×3.. تذكّر أنّ المكعّب الأصليّ الذي أنشأناه كان 1×1×1.

2- تدوير هذا المكعّب حول محور السينات ثمّ محور الصادات بمقدار زاوية الدوران التي حسبناها من قبل.. لاحظ أنّنا قمنا بتحويل الزاوية إلى القياس الدائريّ وذلك بالضرب في النسبة التقريبيّة ط والقسمة على 180:

cube1Angle \* (Math.PI / 180)

لاحظ كذلك أنّ زاوية الدوران تبدأ من صفر حتّى 360 (دورة كاملة) ثمّ تعود إلى الصفر مرّة أخرى.. ولن نحتاج للتدخّل في هذا، فهو من أبسط قواعد الرياضيّات و Direct3D يفهمه!

لاحظ أنّنا لن ننقل هذا المكعّب من موضعه في مركز الشاشة، حيث سنكتفي بتدويره حول مركزه.

تعال نرى ما سنفعله بالمكعّب الثاني:

1- سنغيّر أبعاد المكعّب.. ونحن هنا لن نتركه مكعّبا، بل سنحوّله لمتوازي مستطيلات، طوله وعرضه وارتفاعه مختلفة عن بعضها.

2- بعد هذا سننقل هذا المجسّم بعيدا عن مركز المكعّب الأصليّ بمقدار (-8، -4، 0).

3- ثمّ سندير هذا المجسّم حول محور العينات Z axis بمقدار زاوية الدوران التي حسبناها له من قبل.. إنّ هذا سيجعل المجسّم يدور حول نقطة الأصل.. لماذا؟.. لأنّنا نقلناه من موضعه أولا.. إنّ الدوران سيتمّ حول المحور عين، وليس حور مركز المجسّم نفسه.. ولو أردت أن تجعل المجسّم يدور حول محوره هو، فقم بعمليّة الدوران أولا قبل نقل المجسّم من موضعه.. فهمت الآن فائدة الترتيب؟

ويمكنك الحصول على أنماط حركة مختلفة، بالتلاعب بمعاملات الدوران والانتقال للمكعّبين.. جرّب.

**اعتبارات يجب مراعاتها عند الرسم:**

قبل أن نقوم بالخطوة الأخيرة، يجب أن تضع في اعتبارك أنّ أكثر من 99% من وقت تنفيذ برامج الرسوم يضيع في رسم المجسّمات، وليس في تحميل الصور والخامات ووضع القيم الابتدائيّة.. لهذا فإنّ خبرتك في البرمجة ستتمرّكز حول كيفيّة تحسين أداء تطبيقك، بحيث لا تضيّع وقتا أطول من اللازم في رسم المجسّمات.. وفيما يلي بعض المطبّات التي يجب أن تتلافاها:

**1- الرسم زيادة عن الحدّ Overdraw:**

لو اختبرت أيّ نقطة في الشاشة، فستجد أنّها ترسم مرتين أو ثلاثة للإطار الواحد، نتيجة لأنّ بعض المجسّمات يحجب البعض الآخر، فيتمّ رسم أجزاء من المجسّم الأول على الثاني.. إنّ مخزن البعد الثالث Depth Buffer لا يستطيع حلّ هذه المشكلة، فهو لا يعرف إذا كان هناك مجسّم سيتمّ رسمه مستقبليّا فوق المجسّم الذي يقوم حاليّا برسمه أم لا!.. معنى هذا أنّ هذه مسئوليّتك بالدرجة الأولى، وإن كانت هناك بعض الحلول من Direct3D لا مجال هنا لذكرها.

**2- المجسّمات شبه الشفّافة:**

لكي يعطي Direct3D تأثير الشفافيّة، فإنّه يطبّق معادلات الخلط Blending Equations على المجسّمات الموجودة حاليّا على الشاشة، حتّى تظهر عبر السطح شبه الشفّاف.. فماذا لو تأخّرت أنت في رسم أحد المجسّمات التي يجب أن تكون في خلفيّة السطح شبه الشفّاف؟.. طبعا سيظهر هذا المجسّم المتأخّر عاديّا تماما، دون أن يتأثّر بشفافية السطح الذي أمامه!.. هنا أيضا ترتيب الرسم مهمّ.

**3- الرسم بالقوّة الشرسة Brute-force Rendering:**

يحدث مع حركة المجسّمات أن تخرج من نطاق الرؤية.. فلماذا إذن نضيّع وقتا في رسمها لتشغل حيّزا من الذاكرة وتضيّع الوقت دون أن تؤثّر في شيء؟.. إنّ هناك خوارزميّات لحلّ هذه المشكلة، مثل Frustum culling، occlusion culling، portal culling، BSP Tree’s، وغيرها، ولكنّ المقام لا يتّسع هنا لشرحها!

**رسم المكعّبين على الشاشة:**

آخر خطوة أمامنا الآن هي أن نرسم المكعّبين والقائمة على الشاشة (رسم الإطار).. إنّ ذلك سيتمّ عن طريق الإجراء oneFrameRender.

لاحظ أنّ هذا الإجراء هو المكان الذي يجب أن تكتب فيه حلولا للمشاكل التي ناقشناها سابقا، مثل التخلّص من المجسّمات الموجودة خارج نطاق الرؤية.

ها هو ذا الهيكل العامّ للكود:

**Public Sub oneFrameRender()**

**If Not bInitOkay Then Throw New Exception("…")**

**' محو الإطار السابق ومخزن البعد الثالث**

**D3DDev.Clear(ClearFlags.Target Or ClearFlags.ZBuffer, \_**

**Drawing.Color.FromArgb(255, 0, 0, 64), 1.0F, 0)**

**' بداية الإطار**

**D3DDev.BeginScene()**

**'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**' ضع كود الرسم هنا**

**'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**' إنهاء الإطار**

**D3DDev.EndScene()**

**' تنفيذ رسم المجسّمات**

**D3DDev.Present()**

**End Sub**

والآن فلننتقل لكتابة كود الرسم.. ونظرا لأنّ قائمة التعليمات قد تعلو فتُخفي أجزاء من المكعبين أثناء حركتهما، فيجب هنا أن نراعي الترتيب التالي في الرسم: المكعّبين، القائمة، النصوص التي سنكتبها على القائمة.

سنبدأ بإعداد الجهاز لاستخدام بيانات الرءوس الخاصة بالمكعّب الأصليّ (ممّا سيؤثّر على النسختين المشتقتين منه):

**' تحديد مصدر الرسم عن طريق تمرير مصفوفة رءوس المكعّب**

**D3DDev.SetStreamSource(0, vbCube, 0)**

**' تحديد نوع تنسيق البيانات في مصفوفة الرءوس**

**D3DDev.VertexFormat = CustomVertex.PositionTextured.Format**

**وأخيرا سنرسم المكعّب الأوّل:**

**If bRenderTextures Then**

**' رسم الخامة على أوجه المكعّب**

**D3DDev.SetTexture(0, texCube1)**

**Else**

**' عدم رسم الخامة على أوجه المكعّب**

**D3DDev.SetTexture(0, Nothing)**

**End If**

**' وضع مصفوفة التحويل الخاصة بالمكعّب الأوّل في مصفوفة الموضع**

**D3DDev.Transform.World = matCube1**

**' تحديد طريقة الرسم (قائمة مثلثات)..**

**' يتكوّن المكعّب من 12 مثلثا سنبدأ رسمها من أولها (رقم 0)**

**D3DDev.DrawPrimitives(PrimitiveType.TriangleList, 0, 12)**

لاحظ أنّ bRenderTextures هو متغيّر خاصّ بالخليّة يحتوي على قيمة الخاصيّة useTextures، وهي خاصيّة يجب أن تضيفها للخليّة لتسمح للمبرمج الذي يستخدمها بتحديد إذا ما كان يرغب في استخدام الخامات أم لا (خاصيّة منطقيّة Boolean).. أعتقد أنّك تستطيع أن تكتب كود هذه الخاصيّة، فهو كود تقليديّ لا علاقة له بـ DirectX على الإطلاق.. (راجع الكود في التطبيق المرفق بهذا الفصل).

وسيتمّ رسم المكعّب الثاني بنفس الطريقة، مع استبدال texCube2 و matCube2 بـ texCube1 و matCube1.

الآن علينا أن نرسم القائمة ثنائيّة البعد:

**' حفظ قيمة الخاصيّة في متغيّر احتياطي، لاستعادة قيمتها الأصليّة منه بعد رسم القائمة**

**Dim bPrev As Boolean = wireframe()**

**' يجب تعطيل إمكانيّة رسم الأشكال كشبكة خطوط، لأنّنا لا نريد أن تبدو القائمة كذلك**

**wireframe = False**

**' تفعيل الشفافيّة، حتّى يتمّ استخدام لون الشفافية الذي اخترناه من قبل**

**' إنّ هذا سيزيل اللون القرمزيّ من خامة القائمة، ممّا سيمنحها حوافّ منحنية**

**D3DDev.RenderState.AlphaBlendEnable = True**

**' مصدر الرسم (القائمة)**

**D3DDev.SetStreamSource(0, vbMenu, 0)**

**' تنسيق الرسم**

**D3DDev.VertexFormat = CustomVertex.TransformedTextured.Format**

**' رسم الخامة على القائمة**

**D3DDev.SetTexture(0, texMenu)**

**' طريقة الرسم (شريط مثلثات).. تذكّر أنّ المستطيل يتكوّن من مثلّثين**

**D3DDev.DrawPrimitives(PrimitiveType.TriangleStrip, 0, 2)**

**' إيقاف فاعلية الشفافية بعد رسم القائمة**

**D3DDev.RenderState.AlphaBlendEnable = False**

**' لا تنس إعادة قيمة خاصيّة شبكة الخطوط لقيمتها الأصليّة**

**wireframe = bPrev**

لاحظ أنّ خاصيّة wireframe أيضا هي خاصيّة يجب تعريفها للخليّة، لتسمح للمبرمج بأن يحدّد إذا ما كان يريد رسم المجسّم مغلقا، أم في صورة شبكة من الخطوط.. سأترك لك كتابة هذه الخاصيّة، ولكنّي سأرشدك للجزء الخاص بالرسوم فيها.. إذا اختار المستخدم رسم المجسّم مغلقا، فاستخدم الجملة:

**D3DDev.RenderState.FillMode = FillMode.Solid**

وإذا اختار أن يرسم المجسّم في صورة خطوط، فاستخدم الجملة:

**D3DDev.RenderState.FillMode = FillMode.WireFrame**

وأخيرا، يجب علينا أن نرسم النصوص التي ستظهر في أعلى النموذج.. وفي هذا سنستخدم الوسيلة DrawText الخاصّة بكائن الخطّ.. لاحظ أنّ هذا الكائن خاص بـ DirectX وهو يختلف عن كائن الخطّ الذي نعرفه في VB، والذي ينتمي لفضاء الاسم Drawing.

**fntOut.DrawText( \_**

**"Frame Rate: " + iFrameRate.ToString + "fps", \_ ' النصّ الذي سيرسم**

**' المستطيل يحدد المساحة التي سيتمّ الرسم عليها**

**New Drawing.Rectangle(5, 5 + (2 \* (iFontSize + 6)), 0, 0), \_**

**DrawTextFormat.Left Or DrawTextFormat.Top, \_ ' محاذاة النصّ**

**Drawing.Color.FromArgb(255, 200, 128, 64)) ' لون النصّ**

لاحظ أنّ عليك أن تحدّد مساحة المستطيل لكي يحتوي الخطّ.. لا تحاول أن تجعله أكبر ممّا ينبغي، لأنّ ذلك يهبط بكفاءة تطبيقك.. عامّة يمكن استخدام بعض وسائل GDI التي تناولناها في الفصل 12 (الرسم والتلوين) في مرجع احتراف VB.Net، لمعرفة حجم المستطيل الذي يحتوي الخط.. راجع هذا الفصل.

[www.visualbasicnet.eeran.com](http://www.visualbasicnet.eeran.com/)

**استخدام الخليّة:**

الآن صارت لدينا خليّة جاهزة.. نريد إذن استخدامها.

كلّ ما سنفعله هو أن نستخدم نموذجا لنرى الرسوم عليه.. لن نحتاج لأيّ أدوات على هذا النموذج، وإن كان باستطاعتنا أن نضع ما نشاء من الأدوات (قد يؤدي ذلك لبعض المشاكل).

ابدأ بتعريف متغيّر من الخليّة على مستوى النموذج:

**Private c3DEngine As CSampleGraphicsEngine**

وفي حدث تحميل النموذج أنشئ نسخة جديدة من الخليّة وضعها في هذا المتغيّر:

**c3DEngine = New CSampleGraphicsEngine(Me)**

واضح أنّ هذا خاصّ بالرسم في نافذة.. أمّا إذا أردت أن ترسم على الشاشة بأكملها، فعليك باستخدام الصيغة الأخرى لحدث إنشاء الخليّة.. في هذه الحالة عليك أن تتأكّد أولا من طور العرض الذي يسمح لك به كارت الشاشة.. ها هو ذا الكود:

**Dim iW, iH, iD As Integer**

**If CSampleGraphicsEngine.isDisplayModeOkay(1024, 768, 32) Then**

**iW = 1024 : iH = 768 : iD = 32**

**ElseIf (CSampleGraphicsEngine.isDisplayModeOkay(1024, 768, \_**

**16)) Then**

**iW = 1024 : iH = 768 : iD = 16**

**ElseIf (CSampleGraphicsEngine.isDisplayModeOkay(640, 480, \_**

**32)) Then**

**iW = 640 : iH = 480 : iD = 32**

**ElseIf (CSampleGraphicsEngine.isDisplayModeOkay(640, 480, \_**

**16)) Then**

**iW = 640 : iH = 480 : iD = 16**

**Else**

**' لا يوجد طور عرض متاح!!**

**Throw New Exception("No supported display modes found")**

**End If**

**c3DEngine = New CSampleGraphicsEngine(Me, iW, iH, iD)**

بعد ذلك يمكننا ضبط خصائص الخليّة:

**c3DEngine.backFaceCulling = False**

**c3DEngine.drawFrameRate = True**

**c3DEngine.useTextures = True**

**c3DEngine.wireframe = False**

**bRunning = True**

لاحظ أنّنا لم نشرح هذه الخصائص، ولكن يمكنك الرجوع إليها في كود المشروع.. ليس فيها شيء صعب.

الآن كلّ ما علينا هو أن نكتب جملة تكراريّة يتمّ فيها تحديث الإطارات ورسمها.. سنعرّف أولا متغيّرا على مستوى النموذج لنستخدمه في الخروج من الجملة التكراريّة:

**Dim bRunning As Boolean = True**

لا تنسَ أن تستخدم حدث ضغط الأزرار KeyDown لتجعل هذا المتغيّر False عندما يضغط المستخدم زرّ ESC.

**Do While bRunning**

**c3DEngine.oneFrameUpdate()**

**c3DEngine.oneFrameRender()**

**' يجب استخدام الجملة التالية للسماح للنموذج بالاستجابة لضغطات الأزرار**

**Application.DoEvents()**

**Loop**

وأخيرا سنكتب حدث KeyDown، لنعطي فيه للمستخدم بعض الإمكانيّات:

**Protected Overrides Sub OnKeyDown(ByVal e \_**

**As System.Windows.Forms.KeyEventArgs)**

**Select Case e.KeyCode**

**Case Keys.Escape ' إنهاء الحركة**

**bRunning = False**

**' الانتقال من عرض الأجسام مغلقة إلى عرضها بالخطوط والعكس**

**Case Keys.F1**

**c3DEngine.wireframe = Not c3DEngine.wireframe**

**' استخدام الخامات أو منع استخدامها**

**Case Keys.F2**

**c3DEngine.useTextures = Not c3DEngine.useTextures**

**' محو الأوجه الخلفيّة أو تركها**

**Case Keys.F3**

**c3DEngine.backFaceCulling = Not \_**

**c3DEngine.backFaceCulling**

**' كتابة معدّل رسم الإطارات أو عدم كتابته**

**Case Keys.F4**

**c3DEngine.drawFrameRate = Not \_**

**c3DEngine.drawFrameRate**

**End Select**

**End Sub**

أخيرا... أخيرا تمّ هذا المشروع التعليميّ بحمد الله.. استمتع بتجربته، مع ملاحظة أنّ بإمكانك تطوير هذه الخليّة لاستخدامها في مشاريعك القادمة.

**ما الذي تعلّمناه في هذا المشروع:**

بعد هذا الفصل التعليميّ، يجب أن تكون قد ألممت بما يلي:

1- أهمّ مفاهيم العالم ثلاثيّ الأبعاد.

2- تكوين المشهد باستخدام الرءوس Vertices والمثلّثات.

3- كيف يعمل Direct3D9 بالتوافق مع نظامك.

4- إنشاء المجسّمات يدويّا، بدون الاستعانة بتطبيقات الرسوم المجسّمة.

5- تحميل واستخدام الخامات.

6- إعداد مصفوفات التحويل (تغيير الحجم أو الموضع أو زاوية الدوران)

7- كيفيّة رسم المجسّمات على الشاشة، وأهمّ الاعتبارات التي يجب مراعاتها عند عمل ذلك.

كانت هذه خطوة صغيرة على بداية الدرب.

والله وليّ التوفيق.