بسم الله الرحمن الرحيم

مدخل المبتدئ إلى برمجة الألعاب بإستخدام الـDirectX ٩,٠

الفهرس:

الدرس السادس:

- Vertexe
- Position Vertexes
- Transformed Vertexes

الدرس السابع:

-تحديد الـ Vertexes في الـTransformed
-رسم مثلث بي الـTransformedColored

الدرس الثامن:

-تحديد الـ Vertexes في الـPositionColored
-الـCamera

الدرس التاسع:

-الـ Translation
-الـ Rotation

الدرس العاشر:

-الـScaling
-جمع الـ Matrix للـ Translation و الـ Rotation و الـ Scaling

الدرس الحادي عشر:

-الـVertex Math
-الـVertex Buffer

الدرس الثاني عشر:

-الـTexture
-أنواع الـVertex

الدرس الثالث عشر:

-الـLighting
-الـMesh

الدرس الرابع عشر

تحريك الأشكال
-الـFont

الدرس الأول:

-عن الدورة
-متطلبات الدورة
-تنزيل الـ(SDK) DirectX ٩,٠
-تنزيل الـ٢٠٠٠ Microsoft Visual Studio.Net

عن الدورة:

راودتني فكرة تعلم كيفية تصميم الألعاب منذ زمن ولأكني لم أعرف من أين أبدأ بل وكيف أبدأ , فقلت في نفسي عليك بي المكتبات العامة ومواقع الإنترنت, لأجد بأن المعلومات المتوافرة فيها شحيحة, وتعتمد على معرفة مسبقة بي هذا المجال وبقيت هكذا تائة أجمع اللقيمات وأحاول ترتيب الكلمات إلى أن وفقني الله إلى معرفة بسيطة, لذلك أسميت هذه الدورة بي (مدخل المبتدئ إلى برمجة الألعاب) فهو مدخل للتائة الذي ما زال يبحث عن الجواب لي (أين وكيف.(

متطلبات الدورة

من الأفضل أن يكون القارئي على دراية بالأساسيات فقط من (الجمل التكرارية ,المصفوفات, الجمل الشرطية, وبعض مصطلحات الـ ,(OOP والأهم من هذا كله هي الرغبة في التعلم.

حتى وإن لم تكن الأساسيات بالأعلا واضحة لديك, فلا بأس, فالنقطة التي لا تكون واضحة لديك أثناء سيرنا في خط الدورة, أعلمني بها وإن شاء الله سأوضحها لك .

سنعتمد في الشرح لهذه الدورة على لغة الـ ,C# مع العلم بأن آلية العمل واحدة بكل اللغات.
بمعنى أنك لو إستوعبت آلية العمل فسوف تستطيع وبكل سهولة تطبيق هذه المفاهيم على أي لغة تطقنها.

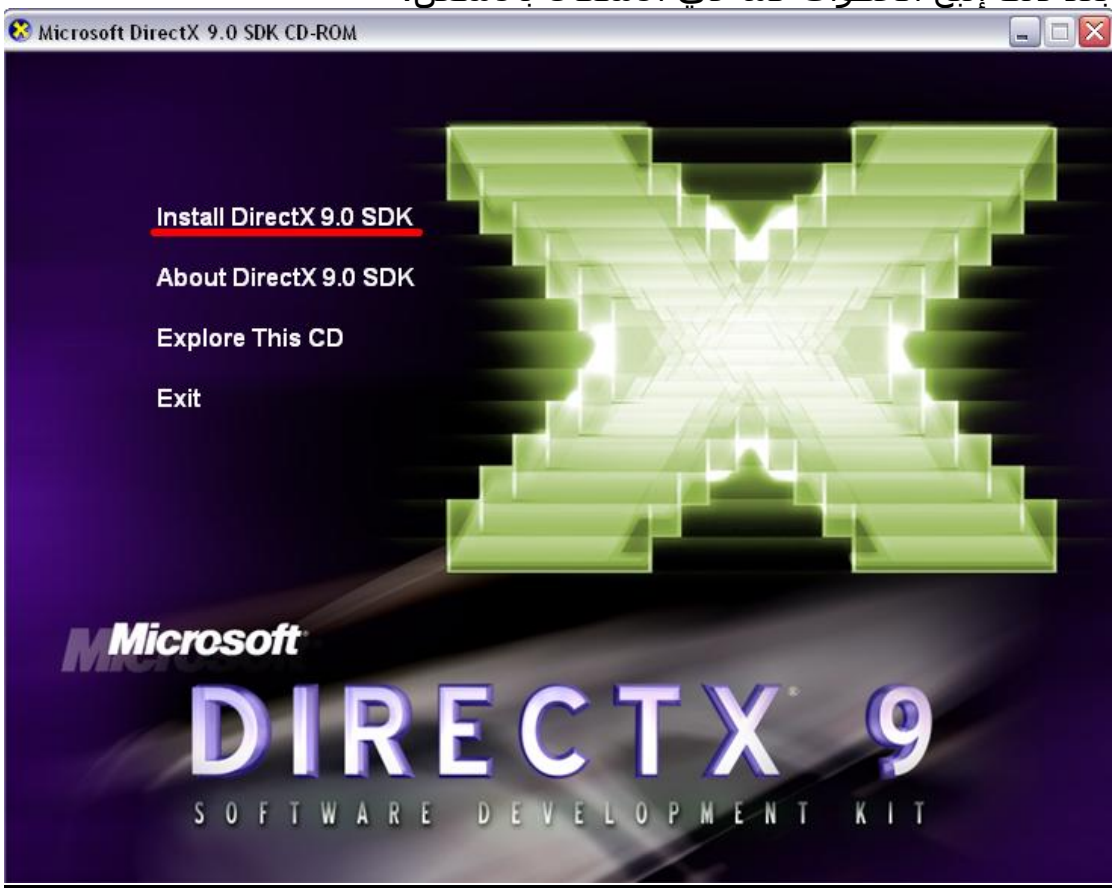
البرامج المستخدمه:

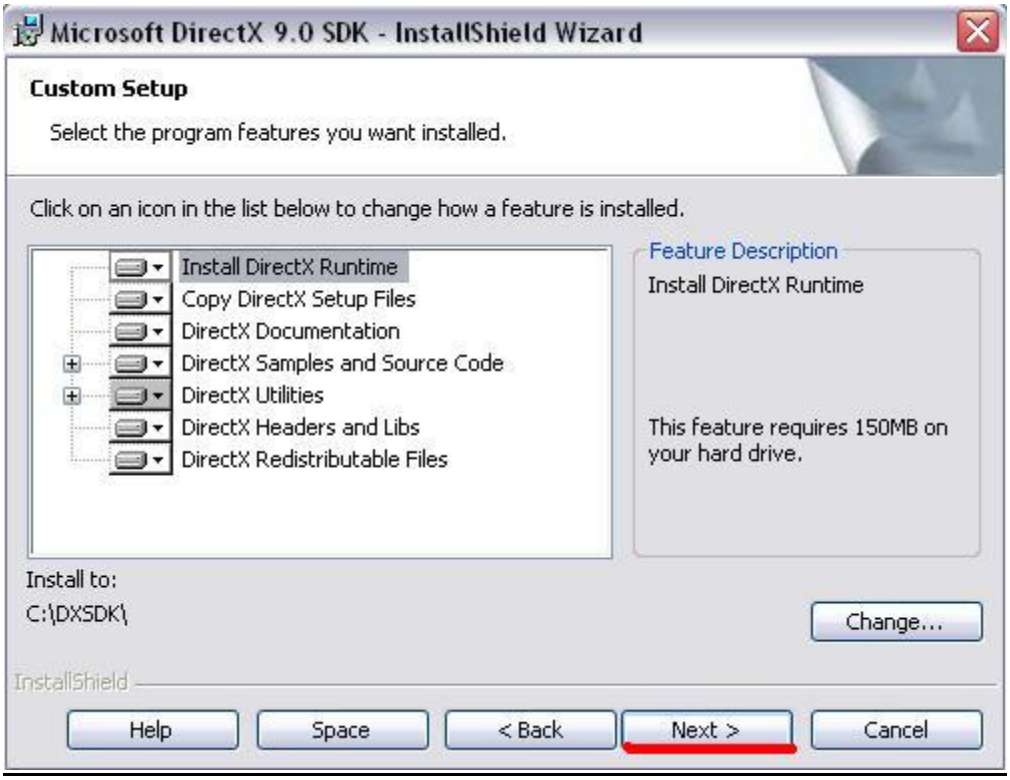
DirectX ٩,٠ SDK (Software Development Kit)

Microsoft Visual Studio.Net ٢٠٠٢

تنزيل الـDirectX ٩,٠ SDK

إذهب إلى الموقع التالي:
http://www.winfuture.de/news,٧٩٥٣.html
وإختر
DirectX ٩ SDK (dx٩sdk.exe, ٢٢٧٧٣٢ KB, englisch)
عند الإنتهاء من تحميل الملف نقوم بفك الضغط عنه.
وبعدها... نفتح الملف ونقوم بالضغط على Install
بعد ذلك إتبع الخطوات كما في الأشكال بالأسفل:





تنزيل الـ٢٠٠٠ Microsoft Visual Studio.Net

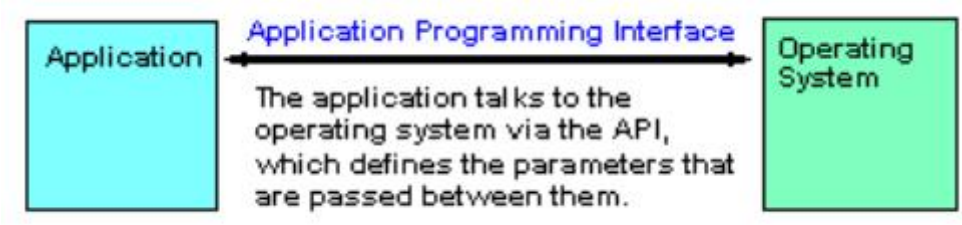
أعتقد أنه لا يحتاج إلى شرح ... فمثل أي برنامج حاول فقط البحث عن الزر (Next)لتصل إلى النهاية.

الدرس الثاني:
-كرت الشاشة
-مكتبات الـAPI
-تاريخ الـDirectX
-مصطلح الـHAL
-مكونات الـDirectX

كرت الشاشة:

بما أن الرسومات تعتمد عليه بشكل رئيسي فكان هو بداية حديثنا, وسنتكلم عنه بإختصار, كرت الشاشة ويطلقون عليه أسماء كثيرة مثل الـ Graphics) Card, Video Card, Video Board, Video Display Board, Display Adapter, Video Adapter, Graphics Adapter), وهي قطعة (كرت) متصل مع اللوحة الأم (Mother Board) تعمل على تخزين ومعالجة الصور وتحويلها إلى إشارات لتستطيع الشاشة إظهارها, أصبحت كروت الشاشة عنصر أساسي في الكمبيوترات منذ عام ١٩٩٠, وأخذ المصنعون يتفنانون بي كيفية صناعاتها حتى توصلوا إلى عمل مجسمات ثلاثية الأبعاد, أول شركة إهتمت بمعالجة الرسومات ثلاثية الأبعاد هي ٣ dfx والتي أطلقت كروت الشاشة الخاصة بي الأجسام ثلاثية الأبعاد وكانت تحل الإسم voodoo وأيضاً هي من إبتكر قناه تسمى بي Graphics Pressing Units (GPUs) وهي منطقة موجودة في كرت الشاشة تقوم بي المعالجات الرياضية للأشكال ثلاثية الأبعاد, مما يريح المعالج (Processor) من هذا العبئ, والذي يؤدي بدورة إلى إسراع عمليات المعالجة للعبه, ومن الممكن أن يحوي كرت الشاشة أكثر من قناة GPU فكلما زادت القنوات إرتفعت الجودة.
من أشهر الشركات في هذا المجال هي:
NVIDIA
Geforce
ATI

مكتبات الـ(Application Programming Interface (API
وهي عبارة عن ملفات يستخدمها المبرمجون من أجل ربط برامجهم بي نظام التشغيل. (Operating System)



تقوم فكرتها كالتالي: بما أن نظام التشغيل يستخدم
بعض العناصر مثل (textbox, *****, font, color, button)
أو بعض الدوال (Functions) مثل (Createwindow, SetwindowText,
ExitWindow, TextOut, Copyfile)
أو بعض المكتبات مثل (User٣٢, Gdi٣٢.dll, Kernel٣٢)
أو بعض من Data Structure مثل (Font)
من أجل التحكم في عملياته
فإذن سأقوم بإستخدامها أنا أيضاً لمصلحة برنامجي الخاص .

ظهرت هذه المكاتب في ثلاثة أجيال وجيل منتظر:
الأولBit ١٦win :

الثاني Bit ٣٢win : حيث كتبت دوالها بإستخدام لغة السي والسي بلاس بلاس, وجائت بالمكتابات(User٣٢, Gdi٣٢.dll, Kernel٣٢)

الثالث Bit ٦٤win :حيث أنها إحتوت على نفس الـ ٣٢win ولأكن بشكل أوسع فتستطيع الدوال الآن معالجة وعنونة أرقام أكبر, وجائت هذه المكاتب على نظامي التشغيل windows XP و ٢٠٠٣ Window Server

الرابع Winfix :وهذا الجيل هو المنتظر في) windows Vista على الأقل إلى الآن) حيث أنها كما أعلنوا ستدعم جميع خصائص الـ Net. بما فيها Garbage Collector System (GLR)وهذة الخاصية خاصة بي إدارة الذاكرة يعني سوف ينسى المبرمجون هذه الإشارة (~) الـ Destructor, وأيضاً سوف تستبدل مكتبات الرسومات وهي الـ GUI بي الـ Avalon والتي تدعم بدورها الخصائص الجيدة لكروات الشاشة مثل المؤثرات (الإضاءة, الظلال), حسب تفسيري الشخصي بأن هذا يعني بأنك من الممكن أن تري ظل الفورم على سطح المكتب, أو صدئ الصوت إذا كان لديك ملف بداخل ملف .

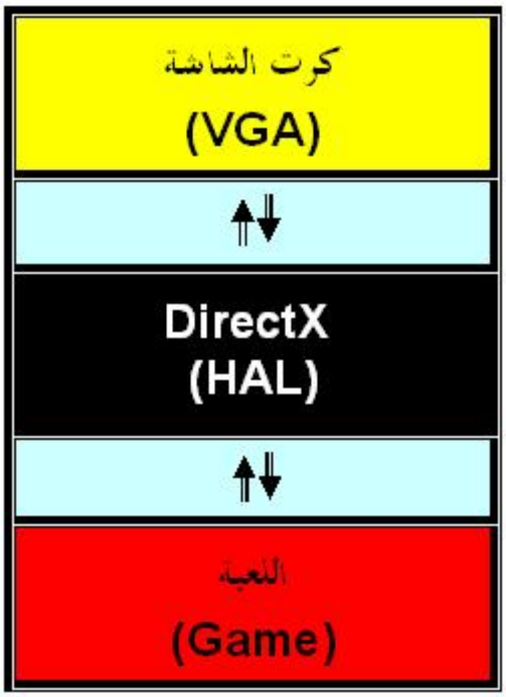
تاريخ الـDirectX

عند التكلم عن برمجة الألعاب فنحن ضمنياً نتكلم عن لغة الجرافيكس (Graphics Language), ومن أمثلة هذه اللغات الـ DirectX والـ OpenGl والـ GDI وتستخدم هذه اللغات في نظام ويندوز و يونكس وأيضاً QuickDrow التي تستخدم على أنظمة ماكنتوش.

يعتبر الـ DirectX جزء من مكتبات API, أول ولادة للـ DirectX كانت في عام ١٩٩٥, وكان في البدء يعمل على النظام DOS, وبقي يتدحرج في عربة التقدم إلى أن وصل إلى النسخة الثامنة والتي نستطيع أن نقول بأنها نقطة التحول فقد إستطاع المطورون أن يحققوا المعادلة الصعبة وهي السهولة والسرعة معاً, وأيضاً جاءت هذه النسخة بي مصطلاحات جيدة في View Point Object و Transformation و Lighting و Texture سنتكلم عن هذه المصطلاحات في وقتها (وكانت هذه النسخة فعالة مع الـ Low Level API, وبعد هذا التقدم الهائل للنسخة الثامنة جاءت بعدها النسخة التاسعة DirectX ٩ لتحوي جميع عناصر النسخ السابقة بالإضافة إلى أنها تدعم الـ ٢٤ Bit وأيضاً Gamma Support, وجاءت لتواكب التقدم مع الـ Net. لتدعم الـ C# و الـ VB.Net.

مصطلح الـ Hardware Abstraction Layer (HAL)

وهي الطبقة العازلة (المتمثله بي DirectX) ما بين الـ Hardware المتمثله بي كرت الشاشة (والـ Software المتمثل بي اللعبة.)



تكمن فائدة هذا الشكل بالأعلا....
بأن المبرمج للعبه لا يهمه أي نوع في كرت الشاشة سوف يعمل به المستخدم وإنما يهتم بأن تكون لعبته تدعم خصائص الـDirectX.
ونفس الحال مع مصمم كرت الشاشة فهو لا يهتم بي أي لغة سيستخدمها المبرمج لعمل اللعبة ,وإنما يهتم بأن يكون كرت الشاشة الذي لديه يدعم خصائص الـDirectX.

بهذه النظرية أصبح الـ DirectX المترجم ما بين كرت الشاشة واللعبة.

وهذا يعني:
أن أي لعبة ستعمل على أي كرت شاشة
وأن أي كرت شاشة سيعمل على أي لعبة
)نظريات أفلاطون (

إلا في حالة أن كرت الشاشة لا يستطيع أن يفهم المترجم وهو ,(DirectX)
وتحدث هذه الحالة عند وجود تأثيرات إضافية في اللعبة ولا يدعمها كرت الشاشة.

مكونات الـDirectX

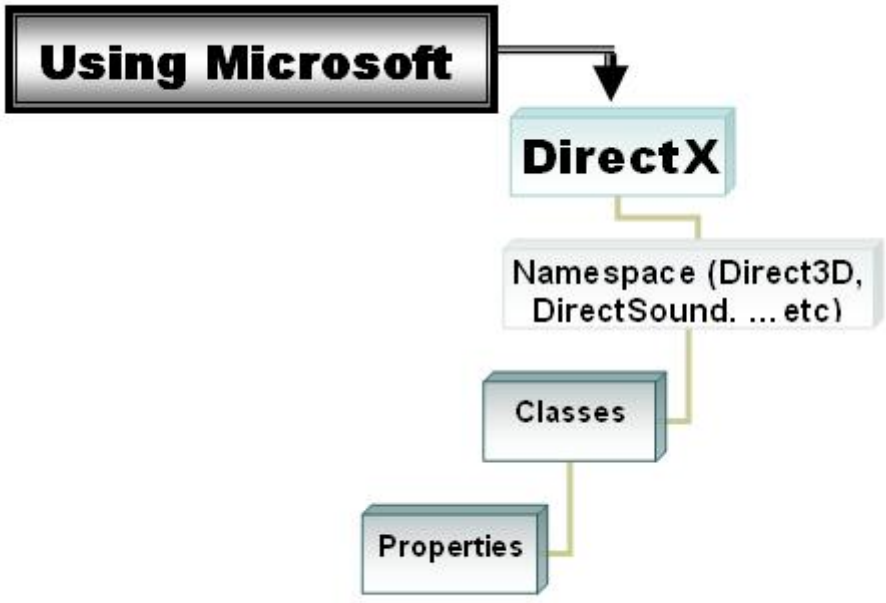
أو تسمى بي الـ (Name Space) سوف نقوم بأخذ فكرة سريعة عن هذه المكونات, وسنتعرض إليها بشئ من التفصيل لأحقاً.
Direct٣D : وهو من أهم العناصر والذي يمثل القالب الذي تتشكل فيه اللعبة (الفورم) وإعطاء الأجسام الشكل ثلاثي الأبعاد (٣.(D
DirectDraw : وهو عنصر مشابه لي الـ Direct٣D ولأكنه يدعم فقط الأجسام ثنائية الأبعاد (٢.(D
DirectSound : وهو العنصر المستخدم من أجل عملية التحكم بالأصوات.
DirectPlay : وهو العنصر المستخدم من أجل جعل اللعبة تدعم أكثر من مستخدم عن طريق الـ .Network بإستخدام.(Client - Server)
DirectInput : وهو العنصر الذي يتيح التحكم بي لوحة المفاتيح أو الفأرة أو الـ Joystick.
AudioVideoPlayback : وهو العنصر الذي يتيح إستخدام الـ Multimedia مثل إدخال ملف فيديو.

الدرس الثالث:
-طريقة التصريح عن الـDirectX
-مقدمة عن الـDirect٣D
-الـPresentParameters
-الـDevice

طريقة التصريح في الـDirectX

لنبدأ بمثال نستخدمه دائماً:
بي الـ C++ من أجل طباعة جملة "You Game Programmer" فنحن بحاجة إلى جملة التصريح وهي include و الهيدر iostream والذي يحوي Cout المختصه بي الطباعة. إذن صرحنا عن الـ iostream من أجل إستخدام المكونات التي بداخله وهي الـCout.

يي الـ C# نفس النظرية نعمل مع الـ DirectX, فنصرح عنه أو عن أحدى مكوناته بإستخدام الجملة using ومن ثم جملة Microsoft وبعدها إسم الـ Namespace (Direct٣D, DirectPlay, DirectSound,.....etc).
أنظر إلى المخطط بالأسفل:



إنها تشبه لعبة الصندوق بداخل ,يجب أن يبقى هذا المخطط في ذهنك التصريح بإستخدام جملة صندوق .. وهكذا إلى أن تصل إلى الهدية, يبدأ

using وبعدها الـ Microsoft وبعدها الـ DirectX والذي يحوي أحد مكوناته وهو الـ Direct3D أو الـ DirectSound .. يحوي عدة الخ والذي بدورة Classes (فئة) والتي بدورها تحوي عدة Properties (خصائص).

البداية مع الـ Direct3D

والرسوم ثلاثية الأبعاد, سنقوم DirectX وهو أول وأهم خطوات التعامل مع الـ بي كتابة الخطوات والمصطلاحات التي سنستخدمها نظرياً ومن ثم سنصيغها برمجي وفي آخر الشرح سنكتب الكود كاملاً بشكل:
Direct3D و الـ DirectX الخطوة الأولى: هي التصريح عن الـ

كود:

```
using Microsoft.DirectX.Direct3D;
using Microsoft.DirectX;
```

ليتعامل مع كرت الشاشة من DirectX الخطوة الثانية: تقوم على تجهيز الـ حيث إدارة وكيفية التعامل معه.
بهذه الخطوة بإستخدام Direct3D يقوم الـ
الـ Class المسمى PresentParameters
الـ Class الأخرى المسمى Device.

الأولى : الـ Class لنبدأ بي الـ PresentParameters

تحوي هذه الـ Class عدة خصائص (Property) منها الـ :
Windowed : وهي الخاصية التي تحدد هل البرنامج (اللعبة) سيعتمد على أم لا,أي أن الفورم Windows Application النوافذ الموجودة في الـ نظام أعلاه زر التكبير والتصغير ولتفعيل هذه الخاصية نعطيها القيمة سيحوي في true. فهذا يعني بأن الشاشة ستكون بهيئة false إعطائها القيمة وعند full screen

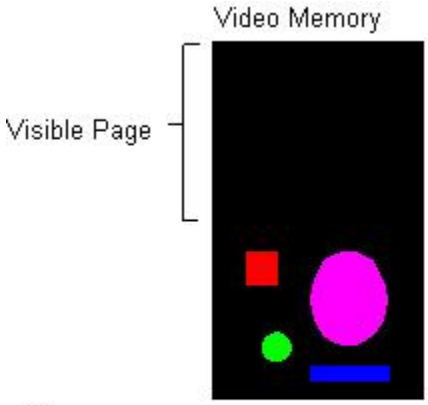
SwapEffect : وهي الخاصية التي تحدد كيفية إظهار الرسومات على الشاشة, تستخدم هذه الخاصية ثلاثة مصطلاحات وهي :

مصطلح الـ Page Flipping:
كود:

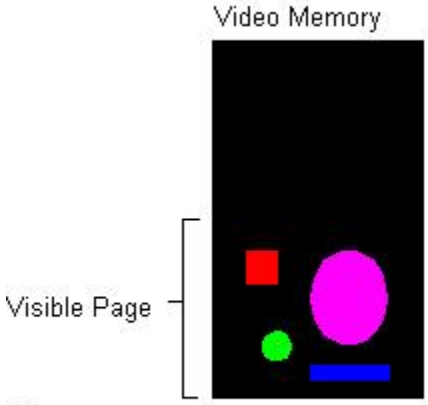
```
SwapEffect.Flip ;
```

عند التكلم عن هذا المصطلح فمعناه بأن كرت الشاشة يحوي مساحة كافية لتخزين مشهدين معاً .(Tow Screen) أو ثلاثة أو أكثر بحسب نوع كرت الشاشة
أي إذا كانت البكسل التي يدعمها كرت الشاشة لدي هي ٣٢٠ x ٢٠٠ ستصبح المساحة الكلية اللأزمة هي ٢ (٣٢٠ x ٢٠٠ = ١٢٨٠٠٠ x ٢)

حيث تنقسم ذاكرة كرت الشاشة إلى قسمين مرئي (أمامي) وغير مرئي (خلفي ,(تبدأ عملية الرسم في صفحة القسم غير المرئي (شكل a (وبعد الإنتهاء تتم عملية تبادل الصفحات لتصبح المعطيات مرئية (شكل ,b (وبعدها تتم عملية تنظيف القسم غير المرئي وتزويده بالبيانات التالية الجديدة.



(a).



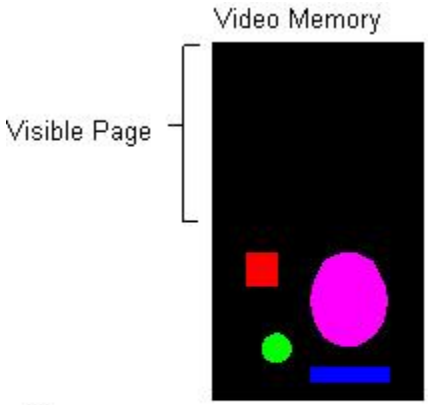
(b).

مصطلح الـ Page Copying:

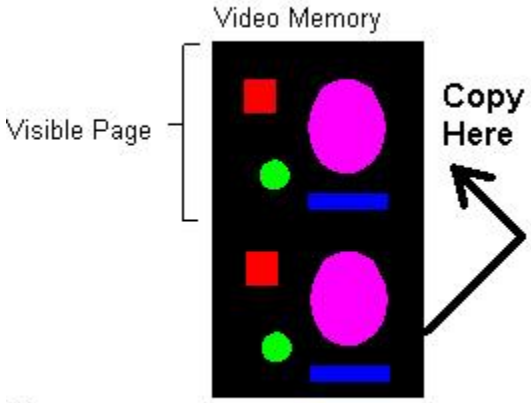
كود:

```
SwapEffect.Copy ;
```

تنقسم ذاكرة كرت الشاشة إلى قسمين مرئي (أمامي) وغير مرئي (خلفي ,(تبدأ عملية الرسم في صفحة القسم غير المرئي (شكل a (وبعد الإنتهاء تتم عملية نسخ (Copy) الصفحات من القسم غير المرئي إلى القسم المرئي (شكل ,b (وبعدها تتم عملية تنظيف القسم غير المرئي وتزويده بالبيانات التالية الجديدة.



(a).



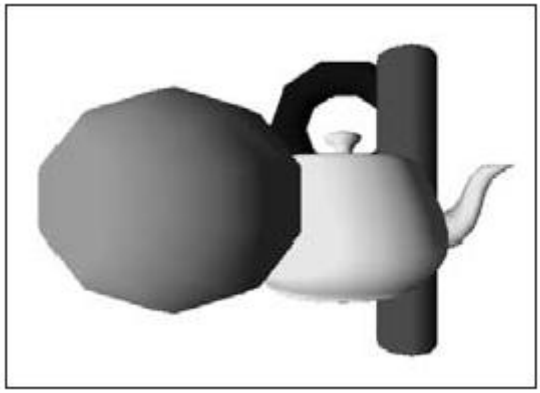
(b).

مصطلح الـ Discarding
كود:

```
SwapEffect.Discard;
```

هذا المصطلح الذي سنستخدمة في دروسنا, حيث يقوم كرت الشاشة بتولي أمر ترتيب وإظهار الصفحات بصورة آليه وبدون تدخل من المبرمج. حيث يمتاز هذا الخيار بالسرعة والدقة, كيف يتم ذلك (على حسب التكتيك الخاص بكل كرت شاشة.(

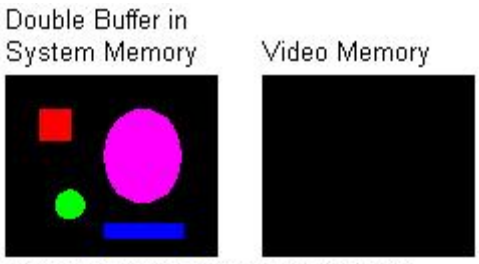
يطلق على هاتين(AutoDepthStencilFormat ,EnableAutoDepthStencil) الخاصيتين بالمصطلح Depth Buffer أو الـ Z Buffer أو الـ W Buffer وهي ذاكرة العمق التي تحدد عمق الأشكال أي أن الجسم القريب يخفي الجسم البعيد ,أنظر إلى الشكل بالأسفل:



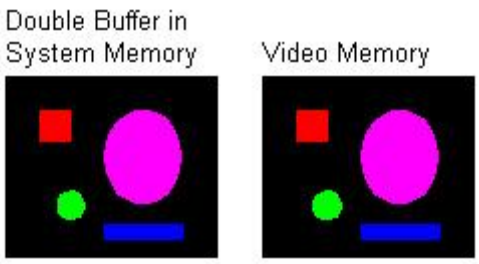
(BackBufferCount, BackBufferFormat , BackBufferWidth, BackBufferHeight)

تستخدم هذه الخصائص من أجل عملية التحكم في الـ Buffer منطقة في الذاكرة تخزن فيها البيانات بشكل مؤقت) من حيث العدد والطول والعرض والنوع .

لنأخذ مثال لتتوضح الفكرة أكثر, عند البدء بالرسم بإستخدام الـ DirectX يقوم يقوم الكمبيوتر بتخزين الأشكال بداخل الـ Buffer الخاص بي الـ RAM أي بداخل جهاز الكمبيوتر (أنظر إلى الشكل ,(a وعند الإنتهاء يقوم البرنامج بنسخ جميع الرسومات إلى ذاكرة (Buffer) الخاص بي كرت الشاشة (أنظر إلى الشكل ,(b ومن ثم يقوم بتنظيف الـ (Buffer) بداخل الكمبيوتر (RAM) ليستقبل البيانات التالية وهكذا....



(a). Instead of drawing to video memory, a double buffer is used.



(b). When finished drawing, the double buffer is copied to video memory.

المراد Buffer بتحديد عدد الـ BackBufferCount حيث تقوم الخاصية Buffer فهو يعتمد على نوع كرت الشاشة لديك وكلما كبر عدد الـ ,إستخدامة كلما زادت سرعة معالجة البيانات. وفي الشكل التوضيحي بالأعلا المستخدم كان الـ BackBufferCount يساوي ٢

حيث أنه يفيد بي Buffer لتحديد نوع الـ BackBufferFormat وتأتي خاصية الـ (Full screen) حالة الرغبة بجعل اللعبة تكون بخاصية ملئ الشاشة

وهما Buffer وتأتي بعدها خاصتي تحديد الطول والعرض للـ BackBufferHeight و BackBufferWidth

ثانياً: الـ Class المسمى الأخرى Device.

مع كرت الشاشة حيث تقوم DirectX كيفية تعامل الـ Class تحدد هذه الـ بي:

أولاً: تحديد أي كرت شاشة سوف يستخدم, نعلم بأن هناك أجهزة تحوي أكثر من كرت شاشة, ويقوم الكمبيوتر بحركة ذكية ليميز أي من هذه الكروت هو الرئيسي (ID Number), فيقوم بإعطاءة الرقم ٠ بنفس هذ الأسلوب هذه الطريقة ليقوم بتحديد أي من كروت الشاشة سوف DirectX يستخدم الـ يستخدم.
ملاحظة: إذا كنت لا تملك إلا كرت شاشة واحد فقط في جهازك فطبعاً سيأخذ الرقم ٠

كود:

```
Device (0, …, …, …, …)
```

ثانياً: تحديد المكان الذي ستتم فيه المعالجة, وتوجد ثلاثة خيارات هم:
أي أن جميع عمليات المعالجة (من إزاحات وإضاءة – سنتكلم Software الـ لاحقاً) ستتم في الكمبيوتر بي وحدة المعالجة المركزية عنها (CPU).
جميع عمليات المعالجة ستتم في كرت الشاشة Hardware الـ (GPU).
أي Debugging يستخدم هذا الخيار فقط من أجل عملية الـ Reference الـ DirectX للـ SDK الإختبار حيث تعتمد عملية المعالجة فيه على عملية

حيث سيعمل على تخفيف العبئ عن Hardware أفضل هذه الخيارات هو الـ الـ CPU.

كود:

```
Device (0, DeviceType.Hardware ,…, …, …)
```

ثالثاً: تحديد الفورم الذي سيعمل الـ DirectX ونستطيع على الرسم عليه,
التعبير عنه بالكلمة (this) أي هذا الفورم الذي نحن فيه.
كود:

```
Device (0, DeviceType.Hardware , this, …, …)
```

رابعاً: تحديد المكان الذي ستتم في معالجة الـ Vectors (وهي النقاط التي يتم رسم الأشكال بواستطها) وتوجد ثلاثة خيارات لذلك هي:

بداخل المعالج Vectors تتم عملية معالجة الـ SoftwareVertexProcessing الـ Central Processing Unite (CPU).
بداخل كرت Vectors تتم عملية معالجة الـ HardwareVertexProcessing الـ الشاشة (GPU), ويجب ان يدعم كرت الشاشة هذه الخاصية وإلا لن تعمل اللعبة.
و الـ CPU تتم عملية المعالجة عن طريق الـ : MixedVertexProcessing الـ GPU.

أو الـ HardwareVertexProcessing لا ينصح بإستخدام الـ MixedVertexProcessing لعدم معرفتنا, هل يدعم كرت الشاشة لي أم لا, برمجياً نستطيع عمل كود للتحقق Vectors معالجة الـ المستخدم قناة يدعم هذه الخاصية كان بها وإلا فإستخدم الخاصية من ذلك فإذا كان SoftwareVertexProcessing سنتكلم عن هذه الأمور في وقتها.

كود:

```
Device (0, DeviceType.Hardware , this, SoftwareVertexProcessing  , …)
```

خامساً: سنضع كائن الـ PresentParameters الأولي وهي الـ Class هذا يعني بأن جميع محتويات الـ Device المسمى Class ستكون بداخل الـ Device (لفهم هذ الكلام يجب أن يكون لديك خلفية بسيطة عن مفهوم الـ OOP).

كود:

```
Device (0, DeviceType.Hardware , this, SoftwareVertexProcessing  ,
Object- PresentParameters )
```

ما رأيك الآن أن نصيغ جميع الكلام بالأعلا برمجياً:

كود:

```
public void ondevice()

                {
```

```
                        PresentParameters pp = new
PresentParameters();
                        pp.Windowed=true;
                        pp.SwapEffect = SwapEffect.Discard;
                        pp.EnableAutoDepthStencil = true;
                        pp.AutoDepthStencilFormat = DepthFormat.D16;

                        device = new Device(0, DeviceType.Hardware,
this, CreateFlags.SoftwareVertexProcessing, pp);


                }
```

لا تقلق سنتكلم بالتفصيل عن مكان وضع هذا الكود وكيفية جعله يعمل ولأكن ليس الآن ... فما يهمنا في هذه المرحلة هو فهم كل عنصر ووظيفته..

كما نرى في الكود بالأعلا فقد قمنا بعمل دالة (function) وأعطيناها الإسم ondevice()وستحوي هذه الدالة الفئتين اللاتين تكلمنا عنهما وهم PresentParametersو الـdevice

الدرس الرابع:
-الدالةOnpaint
-الدالة الرئيسية()Main

الدالة() OnPaint

لتتوضح أهمية هذه الدالة, ما رأيك أن نأخذ مثال عملي: نحتاج إلى كميرا الجوال وشاشة كمبيوتر, الآن إفتح كميرا الجوال لديك ونظر إلى شاشة الكمبيوتر, ماذا ترى؟؟ نعم ما تراه هي الحقيقة لأحظ وجود خطوط في الشاشة مثل ما كأن الخط يقوم بعمل مسح (Scan) للشاشة ويستبدلها بأخرى وتستمر هذه العملية ما دامت الشاشة تعمل.
عملية المسح المتلاحقة أو قل الرسم هي ما تقوم به هذه الدالة, حيث تقوم بعملية عرض الـ Buffer الذي قمنا بتحديد طريقة عرضة عندما تكلمنا عن الـ SwapEffect الموجود في الـ Class المسمى PresentParameters)راجع الدرس الثالث.(

()OnPaintتواجد هذه الدالة في السي شارب في الـ Class التابعة لي Windows Formبشكل .(Built In) وظيفتها الإبقاء على عملية الرسم بشكل مستمر, ولكي تعمل بشكل دائم فلابد من أن تحوي بداخلها حدث (Event)وهو الـPaintEventArgs

لتصبح بحلتها الكاملة كالتالي:

كود:

```
protected override void OnPaint(System.Windows.Forms.PaintEventArgs
                                                                  e)
                                                                   {
                                                                ....
                                                                ....
                                                                   }
```

ما رأيك أن نراجع بعض ما قلنا في الأعلا وفي الدرس السابق لنربط الماضي بالحاضر:
إذن من أعلا .. سيكون على عاتق هذه الدالة (OnPaint) بما تحوية من باراميتر (parameter) التعامل مع الـ Buffer الموجود في كرت الشاشة وتحويلة إلى رسومات.
عملية إظهار الرسومات تكون على شكل: فريم يظهر على الشاشة وفي وقت قصير جدا يبلغ نانو) seconds أي واحد على مليون) يختفي ويظهر الذي يليه وهكذا .. لا أدري هل وضحت الفكرة أم لا ... حسناً سنوضحها أكثر بمثال عملي, أحضر مصباح وقم بفتح الضوء ومن ثم إقفالة .. بسرعة أكبر .. أسرع .. أكثر .. إلى ان تصل إلى سرعة نانو seconds عندها لن تلاحظ بأن الضوء يقفل بل ستراه مضئ بشكل مستمر, وهذا هو نفس نظرية عمل الدالة (Function)المسمى. (OnPaint)

الآن عملية إظهارفريم ومن ثم (مسحه) وإظهار الذي يليه ..... ولأكن لحظة إرجع إلى الوراء من قليلاً ... قلت (مسحه) .. نعم قبل أن يحل الـ فريم الجديد محل القديم فيجب علينا تنظيف المكان, وهنا نأتي إلى موضوع الـ .Clear الذي يجب أن يكون بي هذه الدالة.

تحدث عملية التنظيف (Clear) للـ Buffer وتكون كالتالي:

كود:

```
device.Clear (.. , .., ..,..)
```

هذه الدالة (function) المسمى Clear تحوي بداخلها أربعة عناصر (parameter)وهي كالتالي:

الأول: لتحديد المكان المراد عمل تنظيف (Clear) له وهي

كود:

```
device.Clear (ClearFlags.Target , .., ..,..)
```

ClearFlags.Target :والتي تعني تنظيف الـ Back Buffer أي القسم غير المرئي (راجع الدرس الثالث عندما تكلمنا عن الـ SwapEffect والرسم التوضيحي للقسم المرئي (Front Buffer) وغير المرئي((Back Buffer)

ClearFlags.ZBuffer:والتي تعني تنظيف الـ Z Buffer أي ذاكرة العمق (راجع الدرس الثالث.(

ClearFlags.Stencil : وهو نوع خاص من الـ) Z Buffer ذاكرة العمق) خاص ببعض المؤثرات (Effects) ويجب على كرت الشاشة أن يدعم هذه الخاصية لكي تعمل حيث تختص بأمور الـ.Pixel

الثانية: بعدما إنتهينا من عملية التنظيف يأتي دور اللعب بالألوان, فنقوم بصبغ الـ Buffer باللون الذي نريده ويمكننا إستغلال ذلك بحيث نجعلة لون لخلفية المشهد, نقوم بهذا العمل بمساعدة الدالة Color حيث نقوم بتحديد اللون الذي نريده (أزرق, أحمر, أبيض, أسود ... الخ.(

كود:

```
device.Clear (ClearFlags.Target ,Color.Red , ..,..)
```

الثالثة:
تأخذ رقم ١ وهي خاصة درجة عمق الـBuffer

كود:

```
device.Clear (ClearFlags.Target ,Color.Red , ١,..)
```

والرابعة:
تأخذ رقم ٠ وهي خاصة بي أمور الـPixels
كود:

```
device.Clear (ClearFlags.Target ,Color.Red , ١،٠)
```

دعنا الآن نرتب الكلام الذي وصلنا إليه إلى الآن برمجياً,,, سيكون كالتالي:

كود:

```
protected override void OnPaint(System.Windows.Forms.PaintEventArgs
                                                                    e)
                                                                     {
                                 device.Clear (ClearFlags.Target ,Color.Red , ١،٠)
                                                                     }
```

في الحقيقة كل ما قلناه في الأعلى يبقي مسودة (Draft) لما يجب القيام به إلى أن نصرح عن الدالة (Function) المسمى بي () Present فهي من يحول كل شئ إلى حقيقة لتصبح دالتنا بشكلها النهائي:

كود:

```
protected override void OnPaint(System.Windows.Forms.PaintEventArgs
                                                                    e)
                                                                     {
                                 device.Clear (ClearFlags.Target ,Color.Red , ١،٠)
                                                        device.Present ();
                                                                     }
```

الـ device في الكود بالأعلا هو كائن (object) للـ Device الموجود في الـ DirectX.) سنتكلم عن ذلك في الدرس القادم.(

الدالة الرئيسية()Main

إبدأ من هنا .... هو شعار هذه الدالة فهي نقطة الدخول إلى الكود (Access Point)أو الـ (Entry Point) والتي تقوم بتجميع وإستدعاء الدوال أو الفئات اللازمة, فكل ما كتبنا عنه بهذا الدرس والدرس السابق سنقوم بإستدعائة بواسطة هذه الدالة.

كود:

```
                                                    static void Main()
                                                                {
                                                                …
                                                                …
                                                                }
```

نقوم بأول خطوة بعمل كائن (Object) للفورم لكي أستطيع إستدعاء أي جزء منه بحسب الحاجة, حيث عن طريقة سنقوم بإستدعاء الدالة (function) المسمى ondevice اللأزمة من أجل التحكم في الـ Device التابعة للـ DirectXوالتي تكلمنا عنها في الدرس الثالث.

أما بالنسبة للدالة ()OnPaint فلا يوجد داعي لإستدعائها لأنها كما أوضحنا بأنها. (Built In)

لتصبح كالتالي:
كود:

```
                                        static void Main()
                                                {
                                Form\ xx = new Form\();
                                                {
                                xx.ondeivce ();
                                                }
                                                }
```

إلى هذه المرحلة كل شئ أصبح جاهز, ولأكن بقيت مشكلة واحدة وهي كما تلاحظ بأن الـ Object الذي أنشئناه للفورم ,(xx) يجب أن يستدعي بشكل مستمر ما دام الفورم يعمل لذلك يجب علينا إضافة الجملة التالية لتحقيق هذه الغاية

كود:

```
                                Application.Run (xx);
```

لتصبح الدالة الرئيسية بشكلها الكامل كالتالي:

كود:

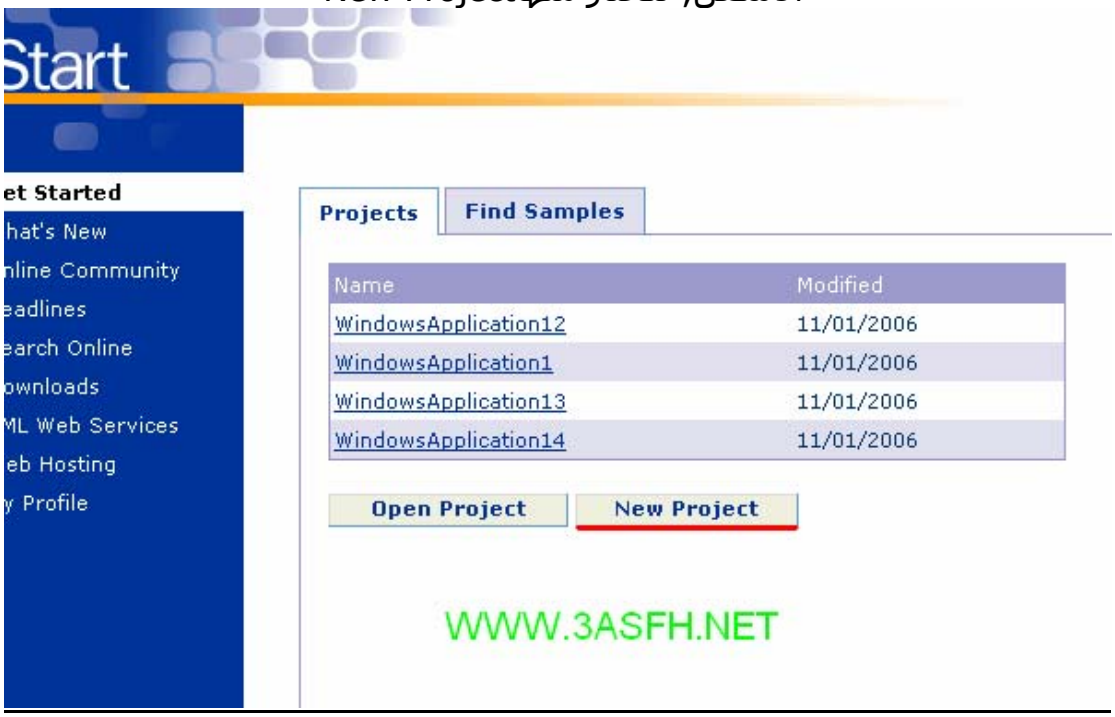
```
                                        static void Main()
                                                {
                                Form\ xx = new Form\();
                                                {

                                xx.ondeivce ();
                                Application.Run (xx);
                                                }
                                                }
```

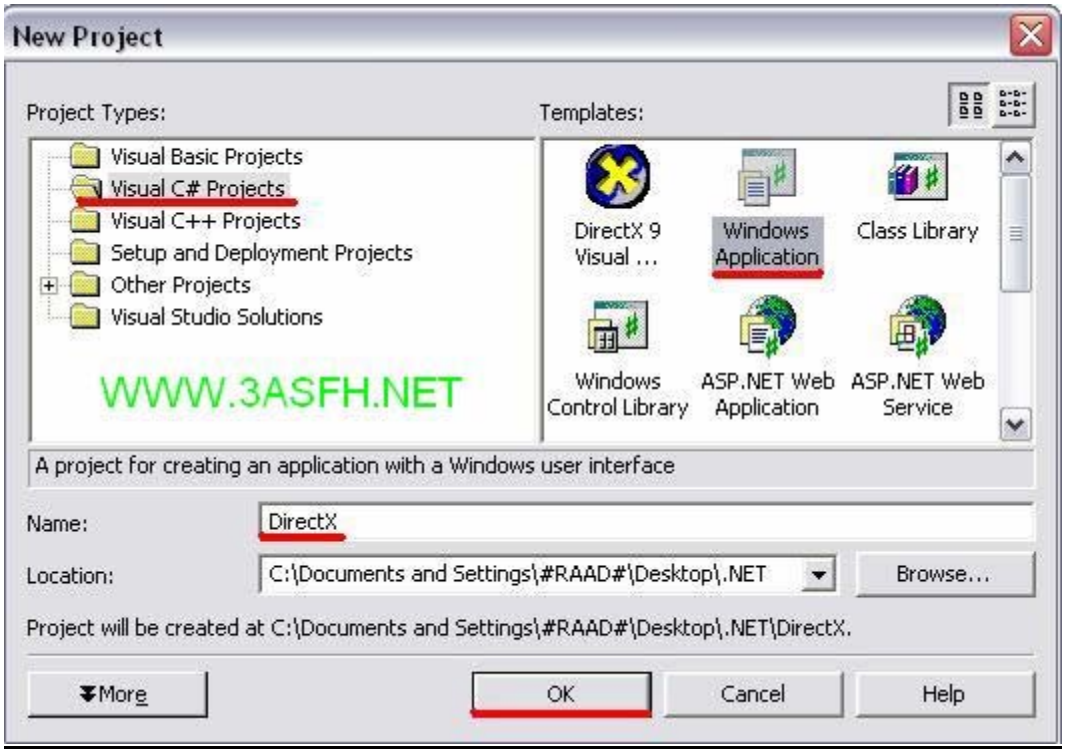
الدرس الخامس:
-تحميل ملفات الـ DirectX في الـ Microsoft Visual Studio.Net
-البداية معWindows Application
-البرنامج الأول مع الـDirectX

تحميل ملفات الـ DirectX في الـ Microsoft Visual Studio.Net

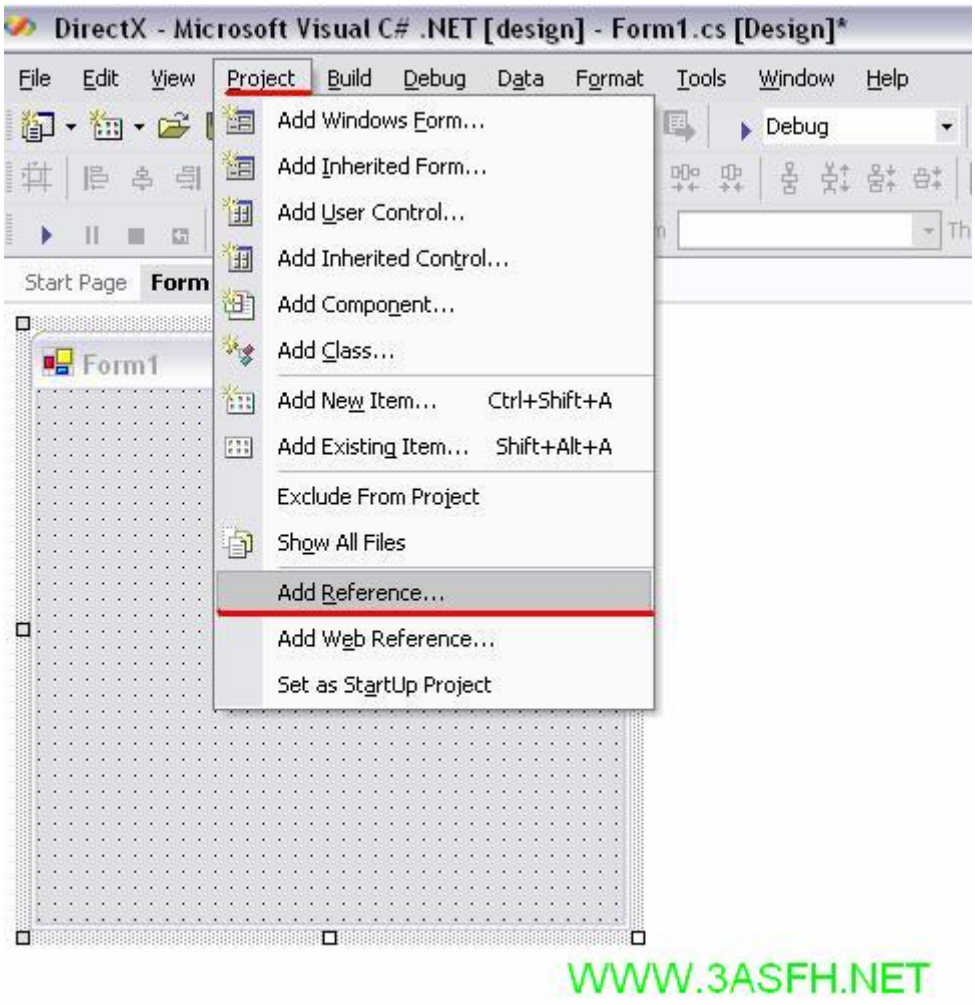
نقوم بفتح برنامج الـ Microsoft Visual Studio.Net لتظهر شاشة كما في الأسفل, لنختار منهاNew Project

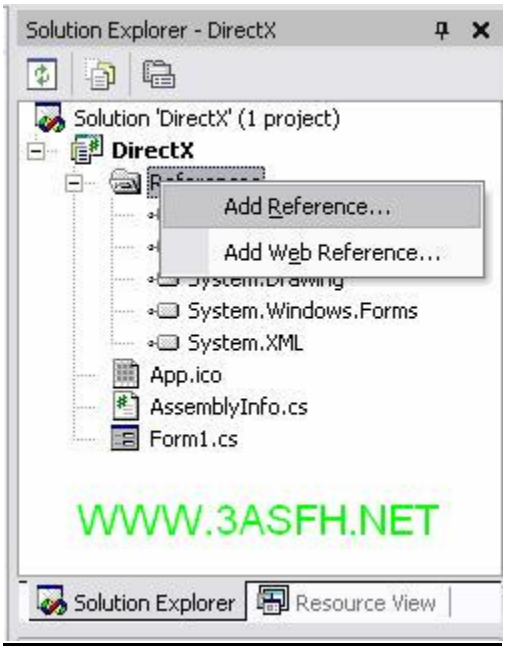


يظهر لدينا الصندوق كما في الأسفل نختار منه Visual C# Projects ومن ثم Windows Applicationوبعدها نختار الإسم الذي نريده للملف وليكن DirectX
وبعدها نقوم بالضغط علىOk

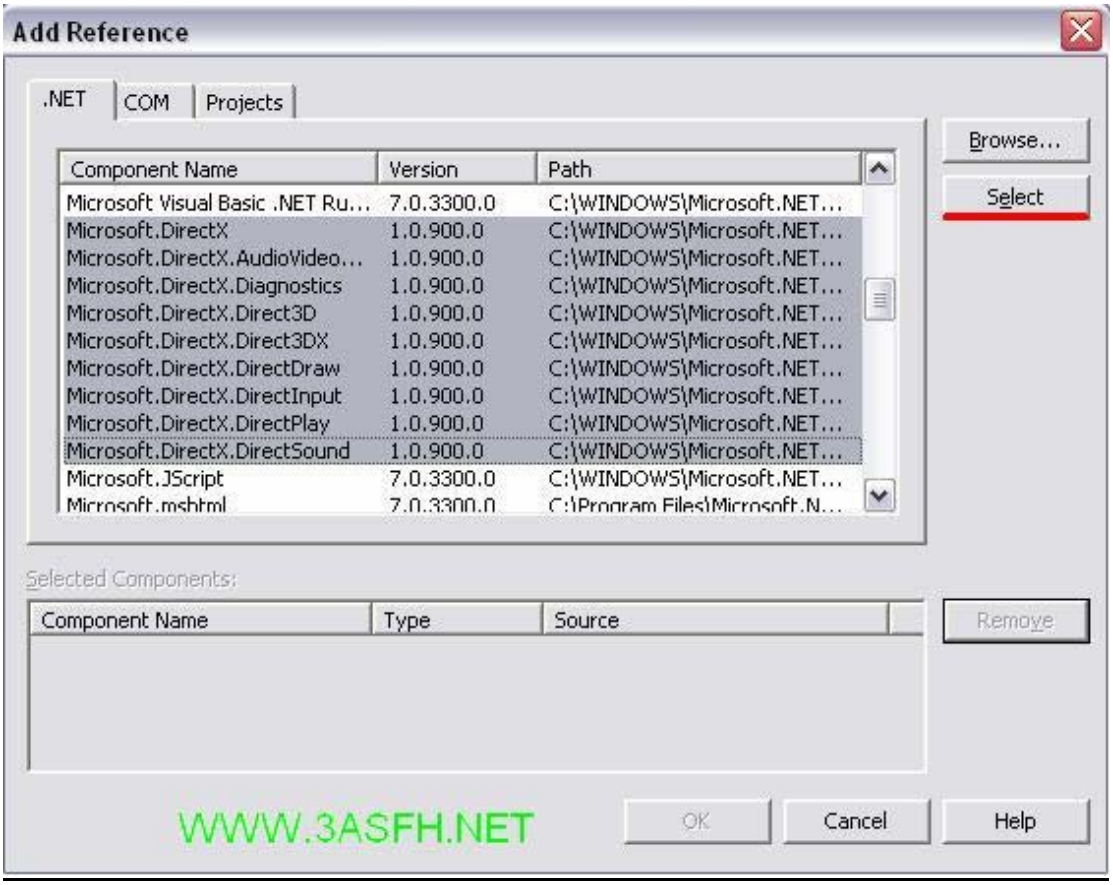


بعدها سوف يظهر الفورم , ونختار من أعلى القائمة Project ومن ثم Add Referenceكما في الشكل بالأسفل

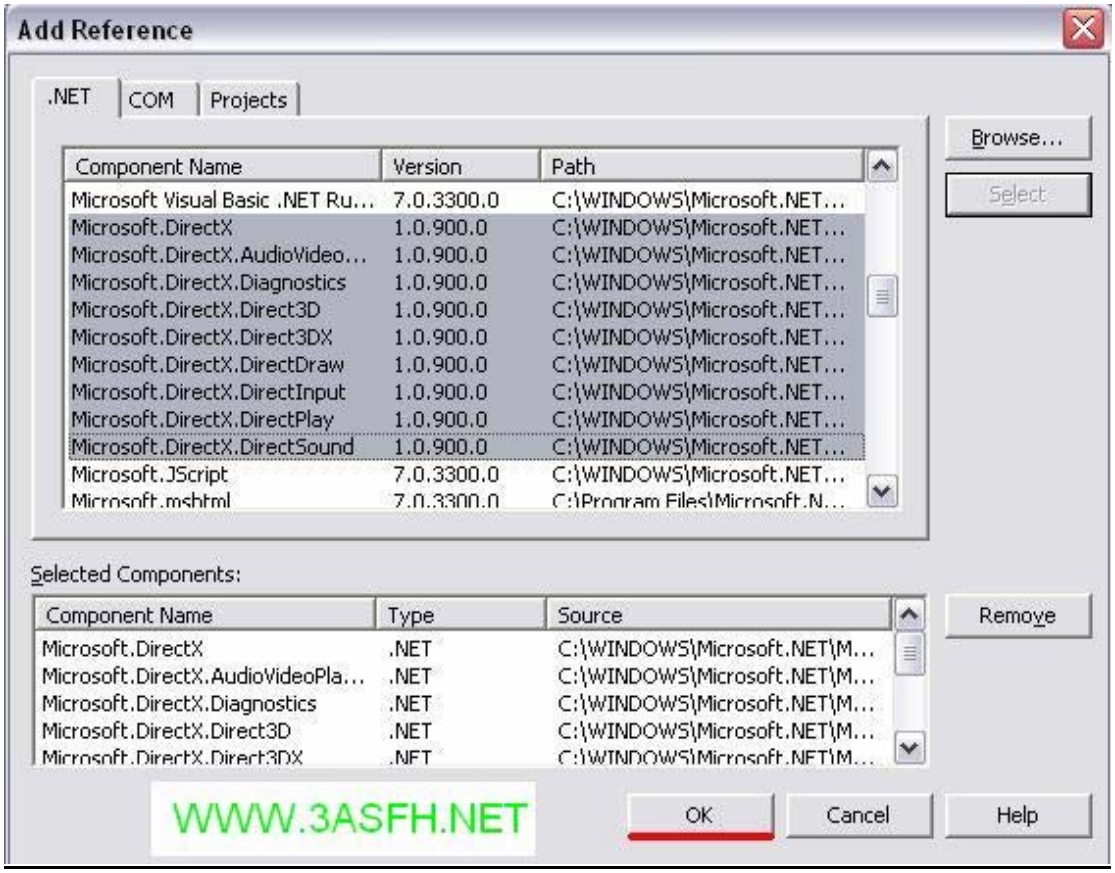




في الخطوة التالية سيظهر لدي الشكل بالأسفل, لنقوم بتظليل جميع مكتبات الـ DirectX, وبعدها نضغط علىSelect



نلاحظ الآن بأن جميع مكتبات الـ DirectX قد ظهرت بالأسفل عند Select Components, كما في الشكل, لنضغط بعدها علىOk

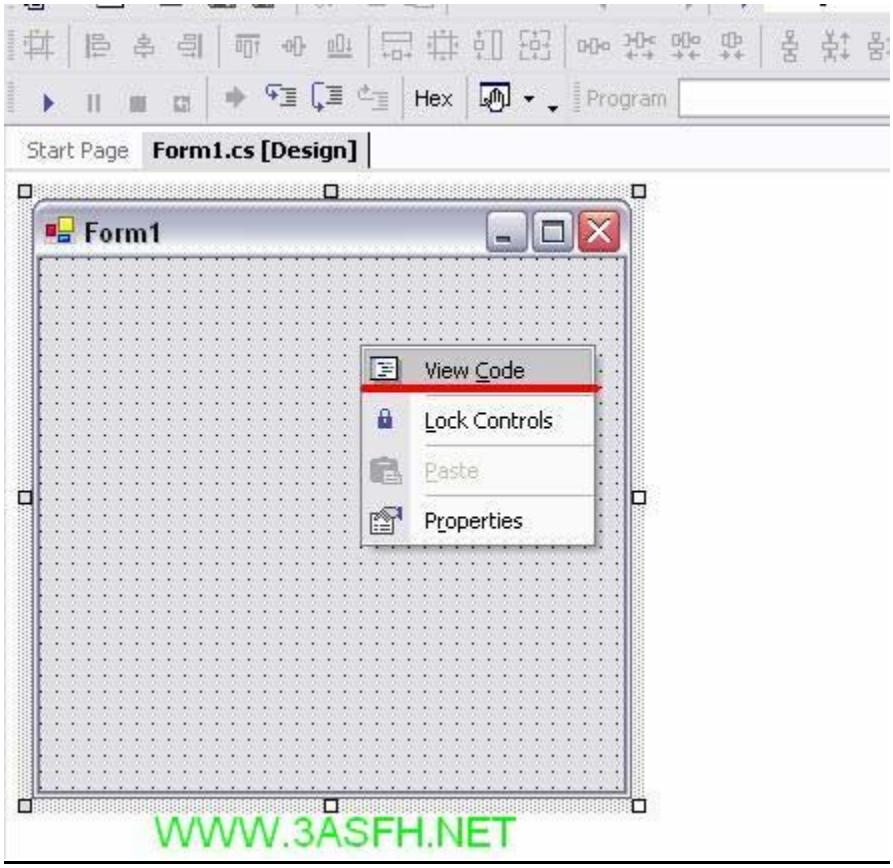


لاحظ الآن بأن جميع مكتبات الـ DirectX قد أضيفت إلى الفورم كما يبين الـ Solution Explorerوهو المربع الموجود على أقصى يمين الشاشة

يجب عليك القيام بهذه الخطوات بالأعلا في كل مرة أردت بها العمل على الـ DirectX

البداية معWindows Application

نقوم بالضغط بالزر الأيمن للماوس (Right Click) على الفور كما في الشكل بالأسفل, ومنه تختار.(View Code)



لنرى الكود في الأسفل, والذي قام الفورم بإنشائة بشكل إفتراضي (On Self Generated).

```
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
namespace DirectX9
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;
        public Form1()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();
            //
            // TODO: Add any constructor code after InitializeComponent call
            //
        }
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
```


(نلاحظ في هذا الكود عدة أشياء, لنأخذ نظرة سريعة: (للعلم فقط

كود:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
```

هذا الكود في الأعلا يمثل الهيدر, وما يهمنا منه الـ

كود:

```
using System.Drawing;
```

والتي تعنى بأمور الألوان.

كود:

```
using System.Windows.Forms;
```

وغيرها من الدوال العامة للتحكم paint وأيضاً هذا الهيدر لأنها تحوي الدالة بي الفورم

كود:

```
public class Form1 : System.Windows.Forms.Form
```

Form و الـ (Form1) ما بين إسم الفئة (Inheritance) قمنا هنا بعمل وراثة من أجل أن يرث جميع خصائص الفورم, وفي هذه المنطقة يتم التصريح عن الكائنات أو المتغيرات.

كود:

```
private System.ComponentModel.Container components = null;

                public Form1()
                {
                        //
                        // Required for Windows Form Designer support
                        //
                        InitializeComponent();

                        //
                        // TODO: Add any constructor code after
InitializeComponent call
                        //
                }

                /// <summary>
                /// Clean up any resources being used.
                /// </summary>
                protected override void Dispose( bool disposing )
                {
                        if( disposing )
                        {
                                if (components != null)
                                {
                                        components.Dispose();
                                }
                        }
                        base.Dispose( disposing );
                }
```

وجود هذا الكود بالأعلا أو عدمه واحد, فهو يختص بي أمور ترتيب الكود وجعل جميع العناصر في قالب واحد (كبر مخك وتركه), لن يفيدنا في دورتنا. وإذا أردت فمسحه.

كود:

```
        static void Main()
```

```
                {
                        Application.Run(new Form1());
                }
```

هنا الحسناء تتربع على العرش ... نعم ها هي دالة (نقطة الدخول) وبها نستدعي أي جزء في الكود .

البرنامج الأول مع الـ DirectX:

في هذا الموضوع سوف نصيغ الدرسين الرابع والثالث برمجياً فهو يعتمد بشكل كبير (وكل شئ سنكتبه في الأسفل قمنا بشرحة في هاذين الدرسين عليهم).

أولاً: نقوم بالتصريح عن الهيدر للـ DirectX كالتالي:

كود:

```
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
```

ثانياً: نصرح عن الـ Device للـ Direct3d
Device device3d;

ثالثاً: نقوم بعمل دالة ونسميها أي إسم ولتكن ondevice() نصرح بداخلها عن الدالتين Device و PresentParameters

كود:

```
public void ondevice()
                {
                        PresentParameters pp = new PresentParameters
();
                        pp.Windowed = true;
                        pp.SwapEffect = SwapEffect.Discard;


                        device3d = new Device (0,DeviceType.Hardware
,this,CreateFlags.SoftwareVertexProcessing ,pp);
                }
```

رابعاً: التصريح عن الدالة OnPaint() ويكون كالتالي:

كود:

```
protected override void OnPaint (System.Windows.Forms.PaintEventArgs
e)
		{
			device3d.Clear (ClearFlags.Target ,Color.Red
,1,0);
			device3d.Present ();
		}
```

خامساً: التصريح عن الدالة الرئيسية ويكون كالتالي:

كود:

```
static void Main()
		{
			using (Form1 xx = new Form1 ())
			{
				xx.ondevice ();
				Application.Run(xx);
			}

		}
```

بهذا نكون إنتهينا من التصريح عن الـ DirectX وعند عمل (F5) RUN سيعطينا الشكل بالأسفل حيث عالم الفضاء, لا تغرك هذه الشاشة الحمراء فهي تخفي ورائها الكثير من المتعة, فهذه ليست شاشة الفورم بل هي شاشة الـ DirectX, وفي الدروس القادمة سترى كيف ترسم وتتحرك الأشكال بداخلها.

الكود كاملاً:

کود:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;


using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;

namespace DirectX9
{
	/// <summary>
	/// Summary description for Form1.
	/// </summary>
	public class Form1 : System.Windows.Forms.Form
	{

		Device device3d;


		/// <summary>
		/// Required designer variable.
		/// </summary>
		private System.ComponentModel.Container components =
null;
		public Form1()
		{
			//
			// Required for Windows Form Designer support
			//
			InitializeComponent();
			//
			// TODO: Add any constructor code after
InitializeComponent call
			//
		}

		public void ondevice()
		{
			PresentParameters pp = new PresentParameters
();
			pp.Windowed = true;
			pp.SwapEffect = SwapEffect.Discard;


			device3d = new Device (0,DeviceType.Hardware
,this,CreateFlags.SoftwareVertexProcessing ,pp);
		}


		protected override void OnPaint
(System.Windows.Forms.PaintEventArgs e)
		{
			device3d.Clear (ClearFlags.Target ,Color.Red
,1,0);
			device3d.Present ();
```

```
		}

		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		protected override void Dispose( bool disposing )
		{
			if( disposing )
			{
				if (components != null)
				{
					components.Dispose();
				}
			}
			base.Dispose( disposing );
		}

		#region Windows Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not
modify
		/// the contents of this method with the code editor.
		/// </summary>
		private void InitializeComponent()
		{
			this.components = new
System.ComponentModel.Container();
			this.Size = new System.Drawing.Size(300,300);
			this.Text = "Form1";
		}
		#endregion

		/// <summary>
		/// The main entry point for the application.
		/// </summary>
		[STAThread]
		static void Main()
		{
			using (Form1 xx = new Form1 ())
			{
				xx.ondevice ();
				Application.Run(xx);
			}

		}
	}
}
```

في صيغة DirectX ما رأيك الآن أن نضيف تغير بسيط بحيث نجعل فورم الـ ملئ الشاشة (Full Screen)
وإضافة الكود التالي windowed حيث سوف نقوم بتعطيل الخاصية:

كود:

```
Format current = Manager.Adapters[0].CurrentDisplayMode.Format;
```

```
                        pp.Windowed = false;
                        pp.BackBufferFormat = current;
                        pp.BackBufferCount = 1;
                        pp.BackBufferWidth = 800;
                        pp.BackBufferHeight = 600
```

والتي تقوم بي تحديد الـ Manager المسمى Class حيت إستخدمنا الـ ID
كما إتفقنا Adapters[0] لكرت الشاشة المراد إستخدامة وهو Number
Screen المراد العمل علية أي الـ (Format) ومن ثم نوع النمط ,سابقاً
Color Quality (٣٢ مثال Bit) بكسل و الـ (769xمثال ١٠٢٤) Resolution..
نوجع رأسنا بهذه الأمور قمنا بتحديد النمط وهو النمط الحالي ومن ثم لكي لا
CurrentDisplayMode.Format عليه المستخدم وذلك بواسطة الـ الذي يعمل

ليصبح الكود بالكامل بالشكل:

كود:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;


using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;

namespace DirectX9
{
        /// <summary>
        /// Summary description for Form1.
        /// </summary>
        public class Form1 : System.Windows.Forms.Form
        {

                Device device3d;


                /// <summary>
                /// Required designer variable.
                /// </summary>
                private System.ComponentModel.Container components =
null;
                public Form1()
                {
                        //
                        // Required for Windows Form Designer support
                        //
                        InitializeComponent();
                        //
                        // TODO: Add any constructor code after
InitializeComponent call
                        //
```

```
		}

		public void ondevice()
		{
			PresentParameters pp = new PresentParameters
();

			pp.SwapEffect = SwapEffect.Discard;

			Format current =
Manager.Adapters[0].CurrentDisplayMode.Format;

			pp.Windowed = false;
			pp.BackBufferFormat = current;
			pp.BackBufferCount = 1;
			pp.BackBufferWidth = 800;
			pp.BackBufferHeight = 600;



			device3d = new Device (0,DeviceType.Hardware
,this,CreateFlags.SoftwareVertexProcessing ,pp);
		}


		protected override void OnPaint
(System.Windows.Forms.PaintEventArgs e)
		{
			device3d.Clear (ClearFlags.Target ,Color.Red
,1,0);
			device3d.Present ();
		}

		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		protected override void Dispose( bool disposing )
		{
			if( disposing )
			{
				if (components != null)
				{
					components.Dispose();
				}
			}
			base.Dispose( disposing );
		}

		#region Windows Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not
modify
		/// the contents of this method with the code editor.
		/// </summary>
		private void InitializeComponent()
		{
			this.components = new
System.ComponentModel.Container();
			this.Size = new System.Drawing.Size(300,300);
```

```
                        this.Text = "Form1";
                }
                #endregion

                /// <summary>
                /// The main entry point for the application.
                /// </summary>
                [STAThread]
                static void Main()
                {
                        using (Form1 xx = new Form1 ())
                        {
                                xx.ondevice ();
                                Application.Run(xx);
                        }

                }
        }
}
```

ولأكن هنا قمنا بحذف الجزء من الكود ... نفس الكود بالأعلا (On Self Generated), لكي تتوضح الرؤيا أكثر ... فيصبح كالتالي:

كود:

```
using System.Drawing;
using System.Windows.Forms;

using Microsoft.DirectX.Direct3D;


public class DX: Form
{
        Device device;


        public void ondeivce()
        {

                PresentParameters pp = new PresentParameters ();

                pp.Windowed = true;

                pp.SwapEffect = SwapEffect.Discard;

                pp.EnableAutoDepthStencil = true;

                pp.AutoDepthStencilFormat = DepthFormat.D16 ;

                device = new Device (0,DeviceType.Hardware
,this,CreateFlags.SoftwareVertexProcessing ,pp);
        }
```

```
        protected override void
OnPaint(System.Windows.Forms.PaintEventArgs e)

        {
                device.Clear (ClearFlags.Target ,Color.SkyBlue
,1,1);
                device.Present ();
        }


        static void Main()
        {
                using (DX xx = new DX ())
                {
                        xx.ondeivce ();
                        Application.Run (xx);
                }
        }
}
```

الدرس السادس:
- Vertexes
- Position Vertexes
- Transformed Vertexes

Vertexes

عند رسم أي جسم فنحن بحاجة إلى حفظ البيانات الهندسية (نقاط المحور xوالمحور y والمحور (z لهذا الشكل, بداخل كائن (Object) يسمى هذا الـ objectبي الـ Vertexes, وبمعنى آخر الـ Vertexes هي النقاط في فضاء الـ DirectXوبواسطة هذا النقاط نرسم الأشكال .
جميع الأشكل في الـ DirectX ترسم عن طريق المثلثات (Rectangle) حتى الدائرة ما هي سوي مثلثات متناهية في الصغر.. أنظر إلى الشكل بالأسفل ليتوضح ما أعنيه.

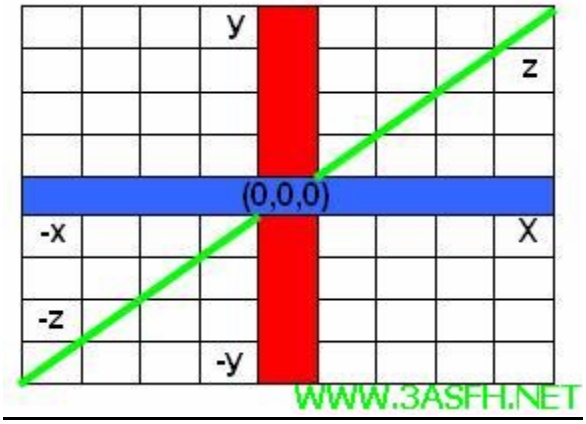
يتيح لنا الـ DirectX بالتحكم في هذه النقاط وذلك بإستخدام الـ Namespace وهي الـ Direct٣D و الـ Class المسمى CustomVertex والتي تحوي بدورها

على نواعين من الـ Vertex هم الـ Position Vertexes و الـ Transformed Vertexes

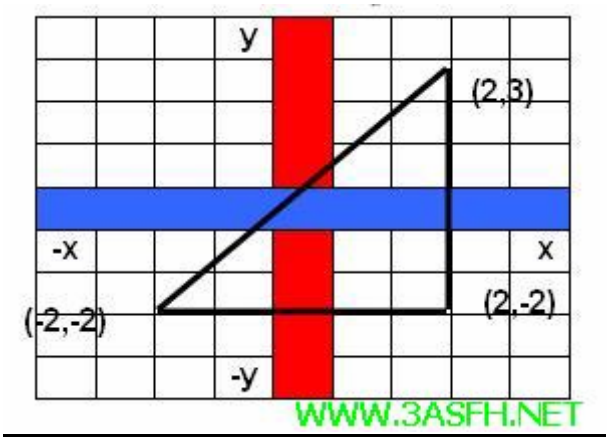
---

Position Vertexes :

يعتمد فيها تحديد نقاط الرسم الـ (Vertex) على محور (x,y,z) حيث أن نقطة البدء تكون في منتصف الشاشة (٠،٠،٠)



ولرسم مثلث فعلينا تحديد النقاط كالتالي:



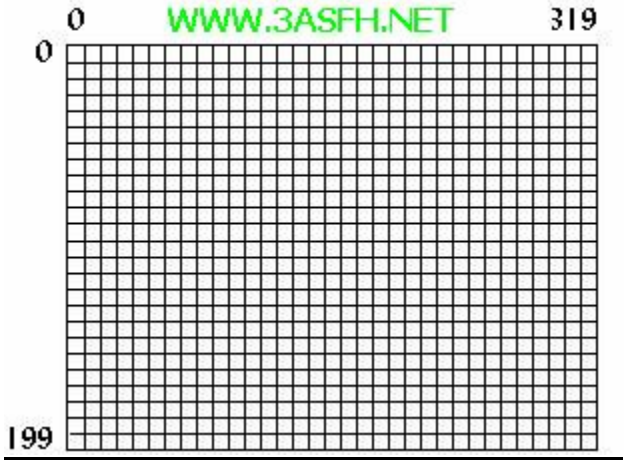
يث يوجد عدة أنواع من الـ Position Vertexes وهي:

PositionOnly: وهو لا يدعم النسيج ولا يدعم الألوان.
PositionColored: وهو لايدعم النسيج ويدعم الألوان.
PositionTextured: وهو يدعم النسيج ولا يدعم الألوان.
PositionNormal: وهو لايدعم النسيج ولا يدعم الألوان.

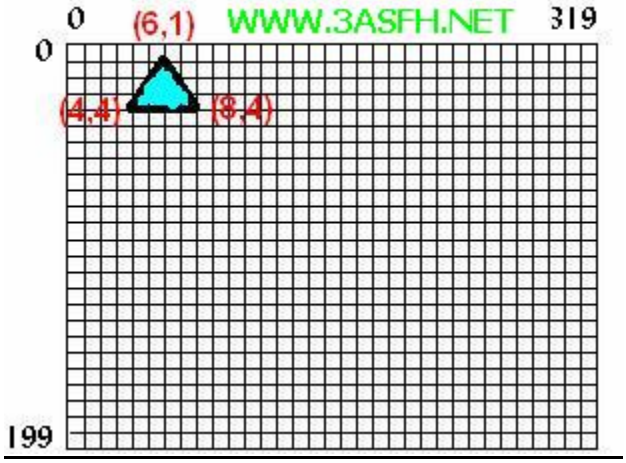
PositionColoredTexturedوهو يدعم النسيج ويدعم الألوان.
PositionNormalColoredوهو لا يدعم النسيج ويدعم الألوان.
PositionNormalTextureوهو يدعم النسيج ولا يدعم الألوان.

Transformed Vertexes :

يعتمد فيها تحديد نقاط الرسم الـ (Vertex) على البيكسل (Pixels) حيث أن نقطة البدء تكون في أقصى يسار الشاشة (٠،٠), لنأخذ المثال التالي :
لنفرض بأن كرت الشاشة لدي يدعم نمط الـ ١٣ hوهو الـ ٣٢٠ ضرب ٢٠٠ (Pixels)فسيكون المحور الذي سنستخدمة من أجل رسم النقاط كما في الشكل بالأسفل.



ولرسم مثلث فعلينا تحديد النقاط كالتالي:



حيث يوجد عدة أنواع من الـ Transformed Vertexes وهي:

Transformed : وهو لا يدعم الألوان ولا النسيج.
TransformedColored: وهو يدعم الألوان ولا يدعم النسيج.
TransformedTextured: وهو يدعم النسيج ولا يدعم الألوان.
TransformedColoredTextured: وهو يدعم الألوان والنسيج.

أقصد بكلمة يدعم/لا يدعم الألوان (Color) بأنه هل من الممكن إعطاء لون للنقاط, وسنتكلم عنه في الدرس القادم.
أقصد بكلمة يدعم/لا يدعم النسيج (Texture) بأنه هل من الممكن إعطاء نسيج للنقاط مثل صورة وسنتكلم عنه في دروس قادمة.

أنظر بالأسفل إلى الجدول الذي يلخص الإختلافات بين الـVertex

Vertex	Transformed	Color	Texture	Normal
PositionOnly	no	no	no	no
PositionColored	no	yes	no	no
PositionTextured	no	no	yes	no
PositionNormal	no	no	no	yes
PositionColoredTextured	no	yes	yes	no
PositionNormalColored	no	yes	no	yes
PositionNormalTextured	no	no	yes	yes
Transformed	yes	no	no	no
TransformedColored	yes	yes	no	no
TransformedTextured	yes	no	yes	no
TransformedColoredTextured	yes	yes	yes	no

هذه هي البداية والدروس القادمة سوف اقوم برفعها انشاء الله
اخوكم هاني العزاوي
Hasa٨٣٨٤@yahoo.com