بسم اللّه الرحمن الرحيم

برمجة الوسائط المتعددة باستخدام DirectX

DirectX 7

البرمجة ثنائية الأبعاد باستخدام Direct Draw.

البرمجة ثلاثية الأبعاد باستخدام Direct3D Rm Mode .

البرمجة ثلاثية الأبعاد باستخدام Direct 3D Im Mode.

برمجة الأصوات باستخدام Direct Sound.

برمجة الموسيقى باستخدام Direct Music.

برمجة وسائل الإدخال باستخدام Direct Input.

مرفق مع الكتاب كافة الأمثلة , والموارد المطلوبة , والوارد ذكرها أثناء سياق الشرح ...

الإهداء

إلى من أعطتني الرعاية دائما

إلى من كانت بجواري دائما

إلى من شجعتني دائما

إليها أقول

لو كان هذا الكتاب يوفي جزء بسيط من حقك فهو لك

لو كان عمري يوفي جزء من حقك فهو لك

لو كانت روحي توفي بجزء من حقك فهي لك

لو أن هذا الكتاب يشعرك بالرضا عني لثانية واحدة

لو أنه يسعدك ثانية واحدة

فإليك

أهدي هذا الكتاب

لـ (أمي)

المقدمة

هذا الكتاب لا يتكلم عن برمجة الألعاب , كما أنه لا يتكلم عن برمجة تطبيقات الوسائط المتعددة , ولكنه يتكلم فقط عن DirectX , والتي تشمل برمجة الألعاب , وبرمجة الوسائط المتعددة , فما هي دايركت إكس ؟؟؟

دايركت إكس عبارة عن مجموعة من الدوال المجتمعة , التي تتيح لك أن تتصل من خلال لغة البرمجة التي تستخدمها , الاتصال بالسواقات المتاحة على جهاز الحاسب , مثل سواقة الصوت Sound Device , وسواقة الفيديو VGA Device , كما تتيح لك الاتصال بوسائل الإدخال مثل Key Board و Mouse و Joy stick , وهذه الدوال مجتمعة مع بعضها في منظومة تسمى مكتبة Library , وهذه المنظومة تسمى DirectX .

كل الألعاب الحديثة , ومعظم التطبيقات الرسومية الحديثة تستخدم DirectX في بناء الصور , أو الأصوات , أو المجسمات , حسب نوع البرنامج , وهذا يبين لك أهمية , وقوة DirectX خاصة إذا علمت أنها تدخل في تركيب معظم البرامج الثلاثية الأبعاد الحديثة مثل 3D Studio Max و Milk shape , كما أن كل الألعاب الحديثة صنعت باستخدام DirectX مثل Fifa2003 .

و DirectX للمستخدم تختلف عن DirectX للمبرمج , أو للمطور , فالمستخدم يراها مجرد ملفات تجعله يستطيع تشغيل الألعاب , والبرامج الثلاثية

الأبعاد , وزيادة جودة الرؤية والصوت , في حين ينظر لها المطور على أنها مكتبات تساعده على تطوير هذه التطبيقات .

دعنا نكتفي بهذه المقدمة , ثم نبدأ في الشرح , ولا تنس أنك بدخولك عالم DirectX فإنك تتحول من مجرد مبرمج عادي إلى مطور تطبيقات وسائط متعددة محترف , وستشعر بالفارق مع الفصول الأولى من الكتاب .

الباب الأول

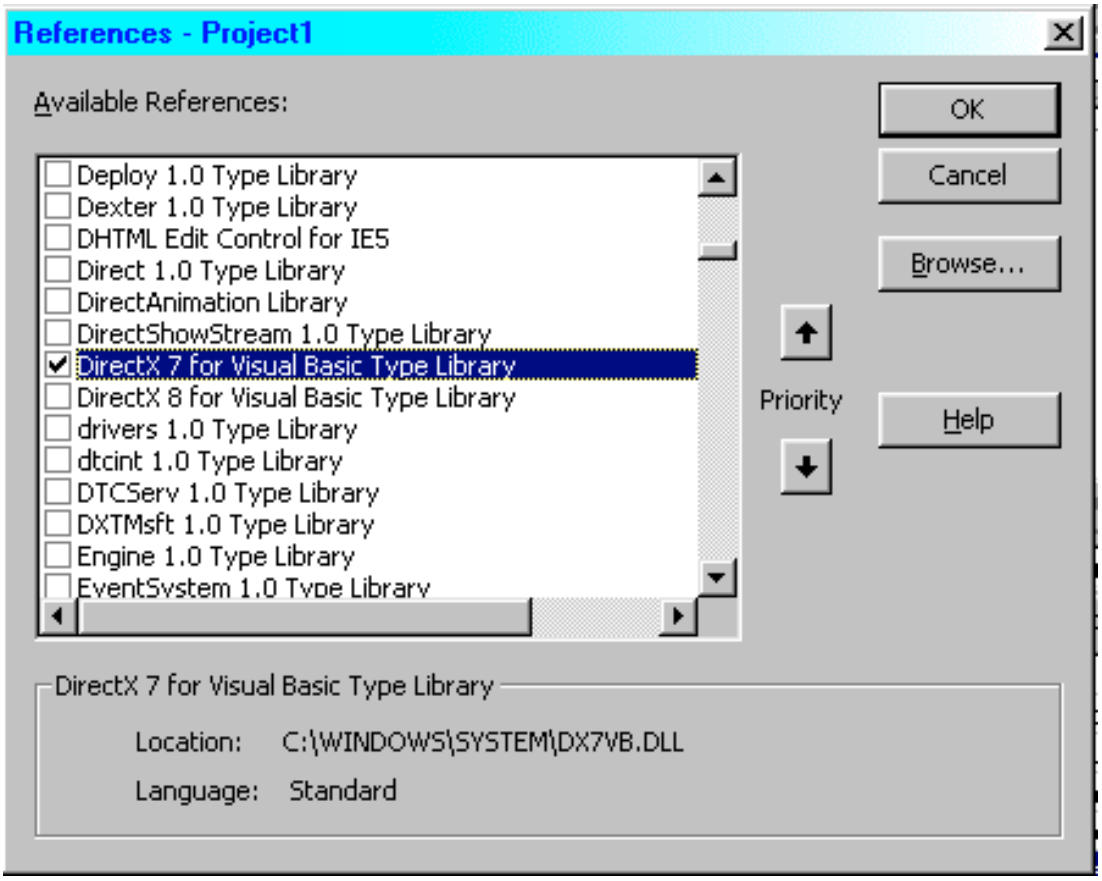
برمجة الرسومات ثنائية الأبعاد باستخدام Direct Draw

على الرغم من أن مجموعة الدوال , والوظائف , والتي تجتمع معا مكونة مكتبة تسمى DirectX , إلا أن هناك كائن أيضا ضمن كائنات هذه المكتبة يسمى كائن DirectX وهو الكائن الرئيسي , أو الوالد Parent , والذي تتفرع أو تتولد منه كائنات DirectX الفرعية ...

قبل كل شيء يجب أن نعلم أن أوامر , ووظائف دايركت إكس ليست متوفرة تلقائيا مع Visual Basic , ولكنها أوامر خارجية , ويجب أن نضيفها من خلال References للمشروع حتى نتمكن من البرمجة باستخدام DirectX .

من قائمة Project داخل بيئة تطوير البيسك المرئية اختر References ,

والتي بالضغط عليها تظهر نافذة الإضافات أو المراجع , ومن هذه النافذة اختر



DirectX 7 For Visual Basic Type Library , كما في الشكل

التالي :

بعد هذا اضغط على OK , ستغلق النافذة , وتعود بك بيئة التطوير إلى نافذتها

الرئيسية ...

اضغط الآن على زر F2 حيث ستظهر لك نافذة مستعرض العناصر Object browser التي فيها تجد جميع أوامر , وثوابت , وعناصر , ودوال DirectX التي يمكن استخدامها من خلال Visual basic.

قبل كل شيء يجب وضع كائن DirectX الرئيسي كالتالي :

```
Dim Dx As New DirectX7
```

الآن نريد أن نتصل بأجهزة العرض لدى المستخدم , لكي نمكنه من اختيار كارت العرض الذي يفضله , لو كان عنده أكثر من جهاز عرض واحد , وأول شيء هو معرفة عدد أجهزة العرض عنده , وذلك باستخدام الكائن DirectDrawEnum والذي يتم تعريفه كالتالي :

```
Dim DE As DirectDrawEnum
```

بعد ذلك نربط العنصر DE بأجهزة العرض لدى المستخدم , وذلك باستخدام العنصر DX كالتالي :

```
Set DE = Dx.GetDDEnum
```

الآن تم ربط العنصر DE بكروت العرض لدى المستخدم , ونحن نريد من كروت العرض هذه شيئين فقط , الأول هو اسم كارت العرض , وهو الاسم الذي سيظهر للمستخدم ليختار كارت العرض الذي يفضله , والثاني هو الرمز GUID وهو الذي سيتيح لنا ربط كائن الرسم الثنائي Direct Draw , بكارت العرض الذي اختاره المستخدم , وسنقوم بتعريف النوعين كالتالي :

```
Dim DeviceNames() As String
Dim DeviceGUIDs() As String
```

الآن نريد أن نعرف عدد محركات العرض لدى المستخدم كالتالي :

```
Dim dCount As Long
dCount = DE.GetCount
```

ثم نعيد تحديد اسم المحرك , ورمزه كالتالي :

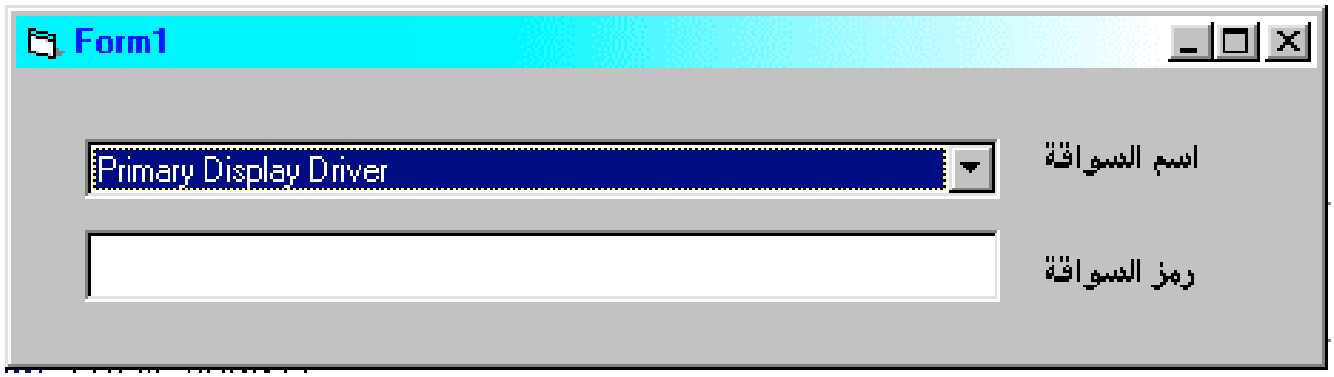
```
ReDim DeviceNames(dCount)
ReDim DeviceGUIDs(dCount)
```

الآن عن طريق حلقة تكرارية من نوع For ... Next سنضع قيم الأسماء , والرموز في المتغيرات النصية الخاصة بذلك , كما سنضع الأسماء التي ستظهر للمستخدم في أداة Combo Box , كالتالي :

```
Dim I As Long
For I = 1 To dCount
DeviceNames(I) = DE.GetDescription(I)
DeviceGUIDs(I) = DE.GetGuid(I)
Combo1.AddItem DeviceNames(I)
Next
```

وفي حدث الضغط على Combo1 نجعل البرنامج يضع GUID المحرك الذي تم اختياره من Combo1 في صندوق نص كالتالي :

```
Text1.Text = DeviceGUIDs(Combo1.ListIndex + 1)
```



والآن سيبدو البرنامج بعد تشغيله كالتالي :

لو لاحظت أن رمز السواقة الأساسية غالبا ما يكون فارغا , وهذا لا يعني أن البرنامج به أخطاء , ولكنه (فقط) رمزه هو مساحة فارغة , أو خالية .

لقد وضعت مع الكتاب مثالا في مجلد الأمثلة باسم المثال الأول , وهو عبارة عن البرنامج الذي صنعناه توا , ليختر منه المستخدم كارت العرض المفضل .
لعلك تتساءل , ما هي فائدة معرفة أجهزة العرض لدى المستخدم ؟؟

والحقيقة أن لها فائدة كبيرة بالنسبة لمن يملكون أكثر من جهاز عرض , كما أنها سيكون لها فائدة لك في المستقبل ...

ولكن الحقيقة هي أن اسم السواقة لا يعنينا في شيء , ولكن يعنينا GUID الذي من خلاله سنختار جهاز العرض الذي سيتعامل معه Direct Draw وهو كائن الرسوميات الثنائية الأبعاد , من كائنات DirectX .

يمكن استخدام Direct Draw مع كلا من النوعين Full-Screen أو Window-Mode , ولكننا من خلال هذا الكتاب سنتعامل مع Full-Screen لأنها تتيح لك استخدام كل إمكانات Direct Draw , وكذلك تتميز بأنها أسرع من Window-Mode .

يستخدم DirectX الرمز GUID في إنشاء كائن Direct Draw , ولما كنا نعلم أن المحرك الرئيسي رمزه مجرد مساحة فارغة , فإن طريقة إنشاء Direct Draw تكون كالتالي :

```
Dim Dx As New DirectX7
Dim Dd As DirectDraw7
Set Dd = Dx.DirectDrawCreate("")
```

لاحظ هنا في السطر الأخير من الكود وضعنا علامتي تنصيص متلاحقتين " " وهذا هو رمز السواقة الرئيسية , ولو كنا نريد استخدام سواقة أخرى علينا أن نستبدل رمزها بالجزء الخالي فيما بين علامتي التنصيص .

لأن Direct Draw يوصلك مباشرة بكارت العرض VGA Card , فإن استخدام Direct Draw يجبرك على تحديد نوع اتصال برنامجك بكارت العرض , وكذلك مدى السماح للبرامج الأخرى باستخدام كارت العرض , بينما برنامجك يعمل ...

ونحن الآن سنجعل برنامجنا يعمل بحجم الشاشة , وهذا يستلزم أن يقتصر التعامل مع كارت الشاشة على برنامج , وبرنامجك فقط , ولكننا سنسمح للمستخدم أن يستطيع استخدام أزرار Alt + Ctrl + Delete في حالة الخروج الطارئ من برنامجك , وكل ما سبق سنفعله في سطر واحد فقط , كالتالي :

```
Dd.SetCooperativeLevel Form1.hWnd, _
DDSCL_FULLSCREEN Or DDSCL_EXCLUSIVE Or _
DDSCL_ALLOWREBOOT
```

أولا حددنا الشاشة الرئيسية التي يستخدمها Direct Draw وهي Form1 , ثم أخبرنا البرنامج بأن يجعل العرض بحجم الشاشة Full-Screen , وكذلك اقتصرنا استخدام كارت الشاشة على برنامجنا باستخدام Exclusive , وأخيرا سمحنا للجهاز بإظهار شاشة End Task عن طريق AllowReboot.

وفي حالة عملنا بنظام Full-Screen يمكننا أن نغير نظام العرض Display Mode , حسب ما نريد في لعبتنا , أو برنامجنا .

وتحديد نظام العرض لا يمكن أن يتم اعتباطا , ولكن يتم على أسس معينة , وبنسب معينة فمثلا يجب أن يكون مقياس العرض إلى الارتفاع يساوي ٣ : ٢ وهذه هي المقاييس المسموح بها :

العرض	الارتفاع
٦٤٠	٤٨٠
٨٠٠	٦٠٠
١٠٢٤	٧٦٨

كذلك عمق الألوان يجب أن يتم تحديده , وهو يمكن أن يكون أحد الأرقام التالية :

٨ أبسط الأعماق , ولا يفضل استخدامه

١٦ أفضل الأنواع ويفضل استخدامه بكثرة

٢٤ ليست كل كروت العرض توفر هذا العمق , لذا يفضل تجنبه

٣٢ يتميز هذا العمق بأنه أكثر وضوحا , ولكنه أيضا يجعل البرنامج أكثر بطئا كما أنه ليست كل كروت العرض تدعمه , وخاصة القديم منها .

وكالعادة , فإن تحديد نظام العرض على الشاشة , وكذلك تحديد عمق الألوان يتم من خلال سطر واحد فقط من الكود كالتالي :

```
Dd.SetDisplayMode 800, 600, 16, 0, DDSDM_DEFAULT
```

الآن اكتمل إعدادنا للشاشة , وتحديد مدى وفعالية تعامل برنامجنا مع كارت العرض , وكذلك مدى السماح للبرامج الأخرى بالتعامل مع كارت العرض , وكل ذلك من خلال فقط الكائن Direct Draw .

وقد وضعت مثالا على تغيير نظام العرض في مجلد الأمثلة باسم المثال الثاني .

دعنا الآن نتوقف قليلا عن التقدم في الكائن Direct Draw وتعال ندرس بعض الكائنات التابعة له , والتي تستخدم في رسم الصور , والأشكال على الشاشة .

كائن RECT

كلمة Rect هي اختصار للكلمة Rectangle أي مستطيل , أو شكلا رباعي الأضلاع , قائم الزوايا , ويستخدم Rect من خلال Direct Draw في رسم الصور على الشاشة , ولتحديد مساحة , وأبعاد Rect يجب علينا تحديد أربعة قيم كالتالي :

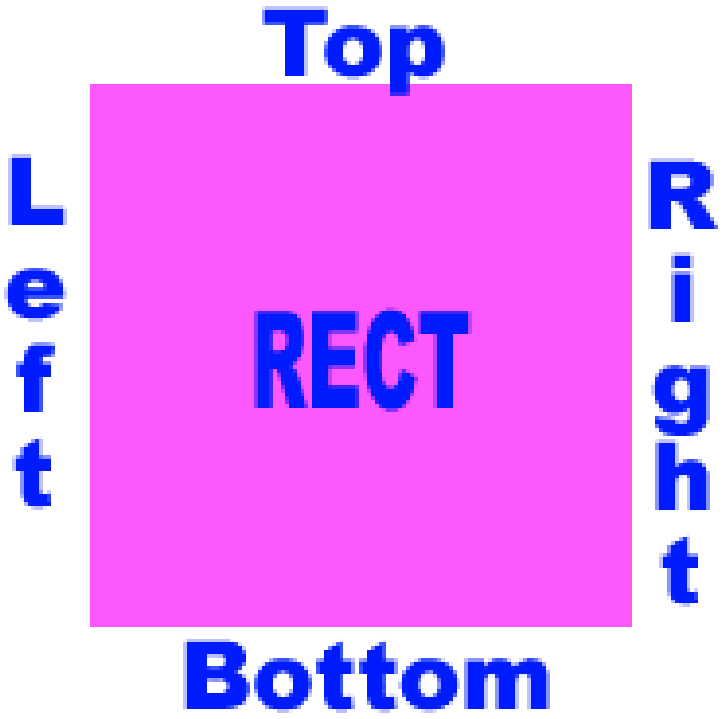
Left بداية المستطيل من جهة اليسار

Top بداية المستطيل من أعلى

Bottom نهاية المستطيل من أسفل

Right نهاية المستطيل من جهة اليمين

ويمكنك أن تفهم أكثر من هذا الشكل التوضيحي :



ويمكن الإعلان عن متغير من نوع Rect كالتالي :

```
Dim R As RECT
```

ويتم تعيين أبعاده كالتالي :

```
R.Top = 20

R.Left = 20

R.Bottom = 430

R.Right = 510
```

كائن DDSURFACEDESC2

يجب أن تعلم أن كائن Direct Draw يعتمد في معظم عمله في إظهار الرسوم على الشاشة على كائن آخر وهو Surface , أي السطح , وهناك ثلاثة أنواع من السطوح تستخدم في الرسم , النوع الأول هو السطح الرئيسي , وهو السطح المسؤول عن إظهار الرسومات على الشاشة , والنوع الثاني هو السطح الخلفي , وهو المسؤول عن أعداد الرسومات قبل ظهورها على الشاشة , والنوع الثالث هو السطح الرسوم , وهو الكائن الذي يتم تحميل الصور التي تظهر في اللعبة عليه ...

وجميع هذه السطوح عبارة عن كائنات من نوع DirectDrawSurface7 فكيف يتم إخبار اللغة أن هذا السطح رئيسي , وذاك خلفي , وهكذا ...

الكائن المسؤول عن تحديد نوع , وصفات , وخصائص السطوح هو الكائن DDSURFACEDESC2 , وهو عبارة عن TYPE مثل RECT تماما , ولا غنى عن استخدامه في التعامل مع السطوح ..

ويتم الإعلان عن متغير من نوع DDSURFACEDESC2 كالتالي :

```
Dim Ddsd As DDSURFACEDESC2
```

قلنا فيما سبق أن السطوح ثلاثة أنواع , سطح رئيسي , وسطح خلفي , وسطح رسومي , فأما السطح الرئيسي , فيتم إعداده كالتالي :

```
Dim Primary As DirectDrawSurface7
Dim Ddsd As DDSURFACEDESC2
Ddsd.lFlags = DDSD_CAPS Or DDSD_BACKBUFFERCOUNT
Ddsd.ddsCaps.lCaps = DDSCAPS_PRIMARYSURFACE _
Or DDSCAPS_FLIP Or DDSCAPS_COMPLEX
Ddsd.lBackBufferCount = 1
Set Primary = Dd.CreateSurface(Ddsd)
```

وأما اعداد السطح الخلفي فيكون كالتالي :

```
Dim Back_Buffer As DirectDrawSurface7
Ddsd.ddsCaps.lCaps = DDSCAPS_BACKBUFFER
Set Back_Buffer = Primary.GetAttachedSurface _
(Ddsd.ddsCaps)
```

أما الدالة GetAttachedSurface فتحتاج لكائن من نوع DdsCaps2 ولكن لما كان DdsCaps2 أحد أعضاء DDSURFACEDESC2 فقد استخدمت Ddsd.DdsCaps بدلا من DdsCaps2 .

ستجد في مجلد الأمثلة , مثالا باسم المثال الثالث , يحتوي على طريقة انشاء السطح الرئيسي , والخلفي .

استخدام السطح الخلفي في رسم الأشكال

يوفر لك السطح (أي سطح) العديد من خيارات الرسم , التي قد تستخدمها في تجميل برنامجك , أو لعبتك , ومن هذه الخيارات :

```
Surface.SetDrawStyle(Long Value)
```

والقيمة من نوع Long تأخذ ارقاما تبدأ من صفر , وهي تتحكم في خط الرسم , كأن يكون خط الرسم متقطعا , أو نقطا , أو نقطة وشرطة , أو غير ذلك .

```
Surface.SetFillStyle(Long Value)
```

والقيمة من نوع Long هنا تتحكم في طريقة ملء الأشكال Shapes باللون , كأن يكون الشكل مليء تماما Solid , أو مملوء بشكل خطوط افقية , أو رأسية , أو مائلة , أو غير ذلك .

```
Surface.SetFillColor (Long Value)
```

والقيمة من نوع Long هنا تستخدم في تحديد لون ملء الرسومات .

```
Surface.SetForeColor (Long Value)
```

والقيمة من نوع Long هنا تستخدم لتحديد لون الرسم , أو الكتابة .

كما يوفر لك عدد من الأشكال التي يمكنك رسمها من خلال Surface كالتالي :

١-رسم صندوق

```
Surface.DrawBox (X1 as Long , Y1 as long , _
X2 as Long , Y2 as Long)
```

٢-رسم دائرة :

```
Surface.DrawCircle (X1 as Long , Y1 as Long , _
R as Long)
```

و R هي قيمة نصف قطر الدائرة , بينما X1 و Y1 هما قيمة موقع مركز الدائرة .

٣-رسم شكل بيضاوي :

```
Surface.DrawEllipse (X1 as Long , Y1 as Long , _
X2 as Long , Y2 as Long)
```

والملاحظ هنا أنها نفس طريقة رسم صندوق .

٤-رسم خط :

```
Surface.DrawLine (X1 as Long , Y1 as Long , _
X2 as Long , Y2 as Long)
```

و X1 – Y1 هما قيمة نقطة بداية الخط أما X2 – Y2 هما قيمة نقطة نهايته .

٥-سم صندوق مستدير الحواف :

```
Surface.DrawRoundedBox (X1 as Long , Y1 as Long , _
X2 as Long , Y2 as Long , rw as Long , rh as Long)
```

و rw و rh هما قيمتي دوران حواف الصندوق .

٦-كتابة نص :

```
Surface.DrawText (X as Long , Y as Long , _
Text as String , b as Boolean)
```

و b تأخذ دائما قيمة False فإذا أعطيتها قيمة True فإن السطح سيتجاهل قيمتي X و Y ويكتب النص في أعلى يسار الصفحة .

وقد خصصت المثال الرابع , في مجلد الأمثلة لموضوع الرسم على السطح الخفي , ومنه ستلاحظ أنه لكي تظهر الرسومات على الشاشة فإنه ينبغي استخدام السطح الرئيسي Primary في اظهار الرسومات على الشاشة كالتالي :

```
Primary.Flip Nothing, DDFLIP_WAIT
```

تعلمنا في الجزء الاسبق كيف نكتب نصا أو (نرسم) نصا على السطح الخفي , ثم نظهره على الشاشة , ولكن كما لاحظنا أن السطح الخلفي لم يمنحنا امكانية التحكم في خط كتابة النص , على الرغم من أهمية هذا , ولكن DirectX لم تهمل هذا , ولكننا نستخدم في هذا كائن من خارج مكتبة DirectX , وهو الكائن StdFont .

أولا نعلن عن الكائن كالتالي :

```
Dim MyFont As New StdFont
```

ثم نحدد صفات الخط المطلوب كالتالي :

```
With MyFont
  .Name = "Arial"
  .Size = 18
  .Bold = True
  .Italic = True
End With
```

بعد هذا نجعل Back_Buffer يستخدم هذا الخط في الكتابة كالتالي :

```
Back_Buffer.SetFont MyFont
```

ثم نستخدم Back_Buffer في كتابة النص كالتالي :

```
Back_Buffer.DrawText 300, 280, "Direct X 7", False
```

وهكذا نرى أن DirectX توفر لك أيضا امكانية اختيار الخط الذي ستكتب به .

وقد خصصت المثال الخامس لهذا الموضوع , من الأمثلة في مجلد (الأمثلة) .

إعداد السطح الرسومي

لعلك الآن تتساءل عن امكانيات DirectX الحقيقية , وأن كل ما رأيته حتى الآن مجرد امكانيات عادية , يمكن تنفيذها بدون الحاجة إلى DX ولكن .. لا تتسرع , هناك المزيد والمزيد من إمكانات DirectX والتي سنتعرف عليها تباعا من خلال الكتاب .

في السطح الرسومي نحن نستخدم أربعة كائنات لكي يمكننا اظهار الصورة من السطح الرسومي إلى الشاشة , الكائن الأول هو كائن DDSURFACEDESC2 والذي من خلاله نعد خصائص السطح , والكائن RECT والذي من خلاله نتحكم

في الجزء الذي سيظهر من الصورة على الشاشة , والثالث هو Back_Buffer الذي يقوم برسم الصورة من السطح الرسومي إلى السطح الخلفي , والرابع هو Primary الذي يتحكم في ظهور السطوح على الشاشة .

والآن سنضع قيم الكائن DDSURFACEDESC2 الذي سنستخدمه لإعداد السطح الرسومي :

```
Ddsd.lFlags = DDSD_CAPS Or DDSD_HEIGHT Or DDSD_WIDTH
Ddsd.ddsCaps.lCaps = DDSCAPS_OFFSCREENPLAIN
Ddsd.lHeight = 600
Ddsd.lWidth = 800
```

لاحظ أننا حددنا قيم الارتفاع Height و العرض Width من خلال الكائن Ddsd , وهكذا نتحكم في حجم الصورة لتي ستظهر على الشاشة .

والآن سنستخدم Direct Draw في تحميل الصورة على السطح الرسومي , ولاحظ أن السطوح في Direct Draw تتعامل مع نوع واحد فقط من الصور , وهو النوع النقطي , والذي يحمل لاحقة (امتداد) باسم *.Bmp :

```
Set Surface = Dd.CreateSurfaceFromFile(App.Path & _
```

```
"\back.bmp", Ddsd)
```

لاحظ أن الدالة CreateSurfaceFromFile تحتاج لمتغيرين , الأول عنوان الصورة , والثاني اسم الكائن DDSURFACEDESC2 المستخدم في تحديد نوع السطح .

الآن سنحدد قيم RECT التي ستكون هنا هي نفس حجم الصورة الذي حددناه مسبقا باستخدام DDSURFACEDESC2 .

```
With R
  .Top = 0
  .Left = 0
  .Bottom = 600
  .Right = 800
End With
```

الآن تم تجهيز السطح تماما للرسم على السطح الخلفي , وهذا هو سطر الرسم :

```
Back_Buffer.BltFast 0, 0, Surface, R, DDBLTFAST_WAIT
```

الدالة BltFast تأخذ ٥ متغيرات هي :

١-قيمة بداية الرسم من جهة اليسار (X) .

٢-قيمة بداية الرسم من أعلى (Y) .

٣-اسم السطح الذي سنرسمه (Direct Draw Surface 7).

٤-المستطيل RECT المتحكم في الرسم (RECT) .

٥-نوعية الرسم .

والآن سنستخدم Primary في اظهار السطح الخلفي على الشاشة :

```
Primary.Flip Nothing, DDFLIP_WAIT
```

وقد وضعت المثال السادس لتوضيح كيفية استخدام السطح الرسومي في مجلد الأمثلة .

استخدام RECT في تقسيم الصورة

حتى الآن نحن لم نر الاستخدام الحقيقي للنوع RECT , حيث أن استخدامه الأساسي يستخدم في تقسيم الصورة إلى أجزاء , والتحكم في الأجزاء التي ستظهر من الصورة , والأجزاء التي لا تظهر .

لنفترض أننا نملك صورة , فيها الحروف من A إلى Z , كالتالي :

ونحن مثلا نريد أن نظهر الحرف T من هذه الصورة , فكيف يتم ذلك ؟؟

أولا نقوم بتحميل الصورة في سطح رسومي , كالتالي :

```
Set Surf = Dd.CreateSurfaceFromFile(App.Path &
"\pic.bmp", Ddsd)
```

بعد هذا نحدد قيمة الحرف T من حيث البعد الأفقي , والرأسي , ولنفترض أن قيمة X لها هي ١٥٠ , أما قيمة Y فهي ٥٠ .

وعلى ضوء هذه القيم نضع قيم RECT كالتالي :

```
R.Top = 50
R.Left = 150
R.Bottom = 100
R.Right = 200
```

والآن نضع أمر الرسم على السطح الخفي كالتالي :

```
Back_Buffer.BltFast 200, 275, Surf, R, DDBLTFAST_WAIT
```

وهكذا نكون قد تعلمنا كيف نستخدم Rect , ولكن لنفترض أنك تريد أن تكتب كلمة كاملة , مثل كلمة DirectX , فهل ستصنع V RECT , وتحدد قيم السبعة ؟؟ أعتقد أن هذا لا يصلح , والحل هو اللجوء لدالة MakeRect والتي بناؤها كالتالي :

```
Function MakeRect(X As Long, Y As Long) As RECT
MakeRect.Top = Y
MakeRect.Left = X
MakeRect.Bottom = MakeRect.Top + 50
MakeRect.Right = MakeRect.Left + 50
End Function
```

وفي هذه الحالة سيكون كود الرسم باستخدام Back_Buffer كالتالي :

```
Back_Buffer.BltFast 200, 275, Surf, MakeRect(150, _
```

```
0), DDBLTFAST_WAIT
```

وستجد المثال السابع يتكلم عن رسم كلمة DirectX في مجلد الأمثلة .

لعل الآن تساؤلك يجري حول تحديد اللون الشفاف من الصورة , بمعنى أن المثال السابع من أمثلتنا كان يكتب حرف T كالتالي :

بينما أنت تريد أن يكتبه البرنامج كالتالي :

بدون الجزء الأسود في خلفية الحرف , وهذا ضروري جدا في تصميم الألعاب , تخيل مثلا أنك تصمم لعبة سباق سيارات , وكانت السيارات تبدو في لعبتك محاطة بمربع , أو مستطيل أسود , أو غير أسود , ستكون لعبتك , بأبسط لفظ يمكن استخدامه (بشعة) .

ولكي نحدد اللون الذي لن يظهر من الصورة نستخدم كائن من نوع DDCOLORKEY , ويكون اعداد الصورة كالتالي :

أولا يتم تحميل الصورة في Surface بالطريقة المعتادة :

```
Set Surf = Dd.CreateSurfaceFromFile(App.Path _
& "\pic.bmp", Ddsd)
```

ثم يتم الإعلان عن كائن من نوع DDCOLORKEY :

```
Dim Key As DDCOLORKEY
```

ثم يتم تعيين قيم الكائن Key :

```
Key.high = 0
Key.low = 0
```

والقيم صفر تعني أن اللون الشفاف من الصورة سيكون هو اللون الأسود , وبعد ذلك نربط السطح بكائن اللون الشفاف :

```
Surf.SetColorKey DDCKEY_SRCBLT, Key
```

وسيتغير أمر الرسم على السطح الخلفي ليصبح كالتالي :

```
Back_Buffer.BltFast 200, 300, Surf, R, _
DDBLTFAST_SRCCOLORKEY
```

حسنا هذا كل شيء , الآن يمكن رسم الحرف T أو أي جزء آخر من الصورة بدون اللون الشفاف منها , ولاحظ أن اللون الشفاف ينطبق على كامل الصورة , بمعنى أنه لو كان هناك جزءا لونه أسود في داخل الصورة فسيختفي أيضا , وليست الأجزاء على الحواف فقط .

وقد خصصت المثال الثامن من مجلد الأمثلة لتوضيح كيفية عمل اللون الشفاف .

<u>إنهاء البرنامج</u>

معظم كائنات DirectX عبارة عن كائنات Classes أي أنك لكي تتعامل معها يجب أن تستخدم اللفظة New , كما في كائن DirectX الرئيسي , أو يتم إنشاءها من خلال كائنات أخرى , كما يتم إنشاء Direct Draw من خلال كائن DirectX , وكائن Direct Draw Surface7 من خلال كائن Direct Draw , وهكذا ...

وهذه الكائنات التي لا يمكن التعامل معها بدون إنشاءها كائنات من نوع Class وهي كائنات تحتل مساحة كبيرة من الذاكرة , ولكن لها مشكلة , وهي أنها تظل عالقة بالذاكرة حتى بعد أن تنهي برنامجك , وقد يتسبب هذا ببطء عمل جهازك , حتى بعد انهاء لعبتك , أو برنامجك .

والحل هو أن تحرر الذاكرة من الكائنات التي استخدمتها في لعبتك , وذلك باستخدام .

```
Set ObjectName = NoThing
```

وهذا كمثال :

```
Set Dx = NoThing
Set Dd = NoThing
```

وهكذا , وهذا من الأمور الواجب عليك عملها دائما حال إنهاء برنامجك .

التحريك في Direct Draw

بالنسبة لـ Direct Draw , فإن الشاشة تنقسم إلى محورين افقي , وراسي , وتتقسم إلى عدد من الطبقات الأفقية , والرأسية , مكونة شبكة , والفارق بين كل خط أفقي , والخط الذي يليه هو قيمة ١ Pixel , ويبدأ ترقيم الخطوط الأفقية من أعلى الشاشة , أي أن أعلى خط هو الخط رقم صفر , ويبدأ ترقيم الخطوط الرأسية من أيسر الشاشة , أي أن أول خط من جهة اليسار هو الخط رقم صفر , في الخطوط الرأسية .

وعلى ضوء ما سبق نرى أن مثلا النقطة (٢٠ , ١٠٠) هي النقطة التي يلتقي عندها الخط الرأسي رقم ٢٠ مع الخط الأفقي رقم ١٠٠ , وهكذا .

يمكننا تحريك الصورة (سطح مثلا) عن طريق تحديد موضعه أثناء رسمه فمثلا أثناء رسم السطوح , فإن أول متغيرين نستخدمهما في عملية الرسم همو الموقع على المحور الرأسي , والأفقي كالتالي :

```
Back_Buffer.BltFast Dx , Dy , ...
```

ولو أننا تحكمنا في نقطة وجود الكائن / العنصر عن طريق تغيير قيمتي Dx و Dy فإننا سنعطي تأثير الحركة .

للتحكم في تحريك عنصر , أي عنصر , أنت تحتاج إلى أربعة متغيرات , تعلنها بنفسك , أولها هو الموقع على المحور الأفقي , ثم الموقع على المحور الرأسي , ثم سرعة التحرك الأفقية , ثم سرعة التحرك الرأسية , كالتالي :

```
Dim X as Long
Dim Y as Long
Dim xSpeed as Long
Dim ySpeed as Long
```

ثم تحدد قيم X و Y كالتالي :

```
X = 100
Y = 100
```

بعد هذا تحدد قيم xSpedd و ySpeed كالتالي :

```
xSpeed = 5
ySpeed = 3
```

وفي الحلقة التكرارية نرسم السطح كالتالي :

```
Back_Buffer.BltFast X, Y, Surf1, _
R, DDBLTFAST_SRCCOLORKEY
```

وفي الحلقة التكرارية أيضا , نكتب أوامر زيادة أو نقصان X و Y تبعا لقيم xSpeed و ySpeed كالتالي :

```
X = X + xSpeed
Y = Y + ySpeed
```

لو أنك قمت بتشغيل برنامجك حتى الآن فسيتحرك السطح في البداية , حتى يصل إلى حدود الشاشة , ثم يختفي بعد ذلك تماما .

وهذا غير مرغوب فيه , لذا فسنقوم ببعض القياسات أولا :

أولا : أثناء تحديد نظام العرض عن طريق Dd.SetDisplayMode حدد أبعاد الشاشة لتكون ٨٠٠ * ٦٠٠

ثانيا : اجعل قيمة ارتفاع وعرض السطح ٥٠ * ٥٠ , وذلك أثناء تحضير Ddsd له , كالتالي :

```
Ddsd.lHeight = 50
```

```
Ddsd.lWidth = 50
```

الآن لدينا كل ما يلزم من جهة القياسات , وبقى أن ننجز جهة الحسابات :

أولا : قيمة ارتفاع العنصر ٥٠ , وعرضه ٥٠

ثانيا : أقصى أبعاد للشاشة هي ٨٠٠ * ٦٠٠

اذن يجب ألا تتعدى قيمة X عن ٨٠٠ – ٥٠ (عرض العنصر) أي ٧٥٠

اذن نضيف الجملة الشرطية التالية في الحلقة التكرارية , وبعد تغيير قيمة X أي بعد السطر الذي كتبناه وهو :

```
X = X + xSpeed
```

نكتب :

```
If X > 750 Then X = 750
```

وننفذ كل ما سبق مع Y مع فارق أن نضع ٥٥٠ بدلا من ٧٥٠ كالتالي :

```
If Y > 550 Then Y = 550
```

الآن نفذ البرنامج , وانظر التغيير .

ولكن لنفترض أنك تريد عكس اتجاه السطح بعد الوصول إلى أطراف الشاشة , أي لو ارتطمت الكرة في طرف الشاشة ترتد متجهة إلى الطرف الآخر , إذن نستبدل جملة IF السابقة بالتالي :

```
If X < 0 Or X > 750 Then xSpeed = - xSpeed
```

وقد خصصت التطبيق الأول من مجلد التطبيقات لتوضيح كيفية تحريك المجسمات في Direct Draw .

التصادم في Direct Draw

لا شك أن أي تطبيق فيه حركة لابد من وجود نظام متكامل لاكتشاف التصادم بين الأجسام , وسواء كان تطبيقك لعبة , أو برنامج , فلابد من أن تتحكم بدقة في حركة الأجسام المتحركة , وإلا تحول تطبيقك إلى فوضى , وعشوائية .

عند محاولة اكتشاف التصادم لابد أن تفكر في المساحة التي يحتلها الجسم الذي تريد أن تكشف التصادم له , ولكي أقرب لك الصورة , تعال نأخذ مثالا واقعيا , فأنت لو كنت مثلا تسير في الشارع , على قدميك , فأنت يمكنك أن تمر على بعد نصف متر , أو أقل بدون أن تصطدم بالرجل إلى جوارك , بينما لو كنت راكبا

سيارة , أو أوتوبيس مثلا , فيجب أن تزيد مسافة الأمان بينك , وبين الناس إلى جوارك .

وهكذا نرى أن التصادم يتم قياسه اعتمادا على حجم الأجسام المتصادمة , لنفترض أن لديك عنصرا مساحته ٥٠ * ٥٠ , وعنصرا آخرا له نفس المساحة , فإن أقل مسافة يجب أن تكون بينهما هي ٥٠ وحدة , ولو قلت المسافة بينهما عن ٥٠ وحدة , يكونان متصادمان ..

مثلا لو أن X1 و Y1 تعبران عن مكان العنصر الأول , و X2 و Y2 تعبران عن مكان العنصر الثاني , فإن اكتشاف التصادم بينهما يكون كالتالي :

```
If X1 > X2 - 50 And X1 < X2 + 50 And Y1 _
Y2 - 50 And Y1 < Y2 + 50 Then
```

الجسمان متصادمان ..

```
Else
```

الجسمان غير متصادمين

```
End If
```

والكود السابق يوضح لنا أنه , لو كان موضع الجسم الأول أقرب من ٥٠ وحدة على المحور السيني وكذلك أقرب من ٥٠ وحدة على المحور الصادي فإن التصادم يتم , والقياس يتم من الجهتين , فعلى المحور السيني يتم القياس من جهة اليمين , ومن جهة اليسار , أما لو كان على المحور الصادي , فإن القياس يتم من أعلى , ومن أسفل ...

وهكذا يتم تحليل حملة `If` السابقة كالتالي :

أولا على المحور السيني :

لو كان مكان المجسم الأول أكبر من مكان المجسم الثاني – المساحة التي يحتلها المجسم الأول , وكذلك أقل من أن يبتعد عنه من الجهة الأخرى مسافة كافية للأمان , فإن التصادم على المحور السيني ...

ثم يتم نفس الاختبار على المحور الصادي , ولو توفرت الشروط الأربعة للتصادم , تم التصادم ...

وقد خصصت التطبيق الثاني من مجلد التطبيقات , لاكتشاف التصادم .

برمجة الكائنات في `Direct Draw`.

برمجة الكائنات , أو البرمجة كائنية التوجه object oriented programming , من الأمور التي لا تكتمل لفظة برمجة بدونها , فلا يمكنك أن تدعو نفسك مبرمجا , بأي لغة برمجة كانت , إلا لو كان لك علم , ومستوى جيد في برمجة الكائنات ...

توفر لك Visual Basic البرمجة الكائنية من خلال الأصناف Class Modules , والتي تتيح لك معظم قدرات البرمجة الكائنية , عدا بعض الوظائف التي لا توفرها Visual Basic مثل الوراثة .

وقبل أن أشرح لك بعض وظائف برمجة الكائنات من خلال Direct Draw , وقبل أن أشرح لك كيفية توظيف الكائن ليساعدك في Direct Draw , فسأعطيك نبذة مختصرة عن الكائنات ...

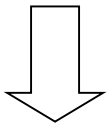
الكائن هو عنصر , أو شيء موجود , له صفات , وله قدرات , وتحدث له أحداث , ويتفاعل معها بإجراءات , وكل شيء في حياتنا يذكرنا بالتوجه الكائني , فالسيارة ككائن , يحدث له حدث , وهو أنك تديرها بالمفتاح , فتقوم بتنفيذ إجراء , وهو إرسال إشارة إلى المحرك فيدور , والمحرك ذاته كائن تابع للسيارة , فما إن تصل له الإشارة حتى ينفذ بعض الإجراءات وهكذا ...

والسيارة ككائن لها صفات , مثل الحجم , والسرعة , والمتانة , وكمية الوقود فيها

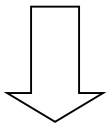
وهكذا تجد أن السيارة تذكرنا ببرمجة الكائنات ...

هذا المثال التوضيحي يوضح لك كيفية عمل الكائنات في البرنامج :

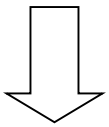
كود عمل البرنامج



أحداث يرسلها الكود إلى الكائن



إجراءات يرد بها الكائن على الكود
بعد أن يتعامل مع الأحداث



متابعة كود البرنامج

وهكذا تجد أهمية وجود الكائنات في لعبتك أو برنامجك , فمثلا لو أنك تعمل لعبة عبارة عن سباق سيارات , فيجب عليك عمل كائن لسيارتك (صنف) , وكائن أو أكثر لسيارات العدو , وكائن للطريق مثلا , أو الحواجز , وهكذا بدلا من أن يكون كودك مفتوحا على بعضه , وتتوه أنت بين سطوره .

سنتخذ معا مثالا على كيفية التعامل مع الكائنات , مع توظيف معظم الشرح على Direct Draw , وكيفية تعامل الكائنات معه , من حيث الرسم , وتغيير طريقة الرسم , واللون وهكذا .

مثال على استخدام الكائنات فى DirectDraw

أولا قم بإنشاء مشروع جديد , وضع فيه فورما , وأكتب فيه كود إنشاء كائنات DirectX كالتالي :

```
Dim Dx As New DirectX7
Public Dd As DirectDraw7
Dim Primary As DirectDrawSurface7
Public Back_Buffer As DirectDrawSurface7
Dim Ddsd As DDSURFACEDESC2
```

وفي إجراء Form_Load تكتب كود تعريف هذه الكائنات كالتالي :

```
Set Dd = Dx.DirectDrawCreate("")
Dd.SetCooperativeLevel Me.hWnd, DDSCL_EXCLUSIVE Or _
DDSCL_FULLSCREEN Or DDSCL_ALLOWREBOOT
```

```
Dd.SetDisplayMode 800, 600, 16, 0, DDSDM_DEFAULT
Show
Ddsd.lFlags = DDSD_CAPS Or DDSD_BACKBUFFERCOUNT
Ddsd.ddsCaps.lCaps = DDSCAPS_FLIP Or DDSCAPS_COMPLEX
_
Or DDSCAPS_PRIMARYSURFACE
Ddsd.lBackBufferCount = 1
Set Primary = Dd.CreateSurface(Ddsd)
Ddsd.ddsCaps.lCaps = DDSCAPS_BACKBUFFER
Set Back_Buffer =
Primary.GetAttachedSurface(Ddsd.ddsCaps)
Back_Buffer.SetFillColor 0
Back_Buffer.SetFillStyle 0
```

كل ما سبق قد قمنا بدراسته بين صفحات الكتاب , والآن سنبدأ في اعداد الكائن , أو الصنف الذي سنقوم بدراسته , وهو عبارة عن رسمة (صورة) , نحدد لها سرعة حركتها , واتجاه حركتها , ثم تحريكها ...

هذا الكائن :

١-صفاته هي : (X – Y – xSpeed – ySpeed).

٢-إجراءاته هي : (الحركة – حدود الحركة (لا يخرج من الفورم) – انشاء الكائن) .

٣- وظائفه : (لا توجد , لأن هذا الكائن لا يمثل أي شيء , ولكنه فقط مجرد وصف عام للكائنات)

الآن من قائمة Project في المشروع , اختر Add Class Module , فسيضيف لك البرنامج أيقونة جديدة , في قائمة Project Explorer باسم Class1 , ويفتح لك في نافذة الكود صفحة جديدة , وفي نافذة الخصائص لن تجد سوى ثلاثة صفات فقط , أهمها هي صفة الإسم , غير الاسم إلى CLASS_TEST .

تضع العناصر الأساسية للكائن كالتالي :

```
Dim Surf As DirectDrawSurface7
Dim Ddsd1 As DDSURFACEDESC2
Dim cX As Long, cY As Long
Dim xSpeed As Long, ySpeed As Long
Dim Key As DDCOLORKEY
Dim R As RECT
```

هذه هي متغيرات الكائن , فأما المتغير Surf فهو السطح الذي سيحمل صورة الكائن , وأما Ddsd1 فستستخدم في اعداد هذا السطح , وأما cX و cY

فسينحكمان في مكان الكائن , و xSpeed و ySpeed تتحكمان في سرعة الكائن , وتستخدم Key في تحديد اللون الشفاف الذي لن يظهر أثناء الرسم , و R هو الـ Rect الذي سيستخدم في رسم السطح .

والآن أول شيء هو إعداد هذه الكائنات , من خلال Sub جديد نسميه بأنفسنا , ويكون اسمه Create كالتالي :

```
Sub Create(Dd As DirectDraw7)


End Sub
```

فأما الكائن Dd الذي استخدمناه في هذا الـ Sub فهو بديل عن Direct Draw الرئيسي الذي أعلنا عنه في الفورم .

وفي داخل هذا الـ Sub نكتب أكواد الإنشاء للمتغيرات كالتالي :

```
Ddsd1.lFlags = DDSD_CAPS Or DDSD_HEIGHT Or DDSD_WIDTH
Ddsd1.ddsCaps.lCaps = DDSCAPS_OFFSCREENPLAIN
Ddsd1.lHeight = 50
Ddsd1.lWidth = 50
```

```
Set Surf = Form1.Dd.CreateSurfaceFromFile(App.Path _
& "\pic.bmp", Ddsd1)
Surf.SetColorKey DDCKEY_SRCBLT, Key
cX = 100
cY = 100
xSpeed = 0
ySpeed = 0
With R
 .Bottom = 50
 .Top = 0
 .Left = 0
 .Right = 50
End With
```

الآن سنبدأ في إعداد صفات الكائن , وهي أربعة صفات فقط , و هي X و Y و xSpeed و ySpeed .

يتم إعداد الصفات بواسطة ثلاثة أشكال :

١ – Property Get : وتستخدم لمعرفة قيمة الصفة .

٢- Property Let : لتعيين قيمة الصفة .

٣- Property set : يستخدم أيضا لتعيين الصفة , ولكن لا نستخدمه في هذا الكائن .

مثلا لو وضعت أداة TextBox في الفورم , و كتبت الكود التالي :

```
Form1.Caption = Text1.Text
```

فأنت بهذا استخدمت Property Get من الأداة Text1 , وكلمة Property تعني صفة , وكلمة Get تعني إحضار , وبالتالي فإن Property Get تعني إحضار الصفة , ومعناها معرفة قيمة الصفة ...

أما لو كتبت السطر التالي :

```
Text1.Text = “Text”
```

فأنت بهذا استخدمت Property Let , الخاصة بالأداة , لتغيير قيمة النص , وكلمة Let تعني (أودع) , ومعناها العام تغيير , أو إيداع قيمة للصفة , فهي تكون تغيير لو كانت هناك قيمة أصلية وغيرتها , وتكون إيداع لو لم تكن هناك صفة

أصلية , وغيرتها , وفي حالة TextBox تكون تغيير وليس إيداع , لأن الأداة لها صفة أصلية وهي : Text1 .

نعود إلى الكائن الذي نقوم بتصميمه , ونبدأ في إنشاء الصفة Y , وهي صفة تعبر عن مكان الكائن على المحور ص , أو المحور الرأسي

نعرف أن الخاصية Y هي مجرد واجهة للكائن , بالنسبة للبرنامج , أم المتغير المتحكم في مكان الكائن فعليا فهو المتغير cY الذي أعلنا عنه فيما سبق .

نبدأ في إنشاء الصفة , بكتابة الكود التالي :

```
Property Get Y() As Long


End Property
```

وبما أن Y الفعلية للكائن هي cY فسنكتب التالي في داخل الصفة :

```
Y = cY
```

المفروض الآن لو كتبت من داخل كود البرنامج , وليس الكائن , التالي :

```
Print Object.Y
```

فسيطبع البرنامج القيمة cY للكائن , ولكن لا تحاول ذلك الآن , لأنه لا يوجد كائن باسم Object حتى الآن , ونحن لم ننشئ الكائن حتى الآن .

ثم ننشئ الجزء الخاص بالإيداع من الصفة , والذي نستخدمه لتغيير أو لإيداع قيمة الصفة , كالتالي :

```
Property Let Y(Value As Long)

End Property
```

وبالطبع فإن القيمة الفعلية هي التي سيتم تغييرها , كالتالي :

```
cY = Value
```

وبهذه الطريقة يتم إنشاء الصفات X و SpeedX و SpeedY , كالتالي :

الخاصية X :

```
Property Get X() As Long
X = cX
```

```
End Property

Property Let X(Value As Long)

cX = Value

End Property
```

الخاصية SpeedX:

```
Property Get Speedx() As Long

Speedx = xSpeed

End Property

Property Let Speedx(Value As Long)

xSpeed = Value

End Property
```

الخاصية SpeedY:

```
Property Get Speedy() As Long

Speedy = ySpeed

End Property

Property Let Speedy(Value As Long)
```

```
ySpeed = Value

End Property
```

الآن سنضع الوظيفة الخاصة برسم الكائن على السطح الخلفي , بأن ننشئ وظيفة خاصة بالرسم باستخدام Back_Buffer , ويتم إنشاء الوظيفة كالتالي :

```
Sub Draw()


End Sub
```

وفي داخل الـ Sub يتم وضع أمر / أوامر الرسم :

```
Form1.Back_Buffer.BltFast cX, cY, Surf, _

R, DDBLTFAST_SRCCOLORKEY
```

والآن سننشئ وظيفة التحريك هكذا :

```
Sub Move()

cX = cX + xSpeed
```

```
If cX > 750 Then cX = 750

If cX < 0 Then cX = 0

cY = cY + ySpeed

If cY > 550 Then cY = 550

If cY < 0 Then cY = 0

End Sub
```

لا تحتاج الوظيفة للشرح , وقد تم شرح كل الأوامر التي ذكرت فيها , في دروس سابقة .

الآن نعود مرة أخرى إلى البرنامج , وفي كود الفورم , في قسم الإعلان العام , أعلن عن الكائن الذي أنشأناه كالتالي :

```
Dim Obj As New CLASS_TEST
```

الآن أصبح للكائن الذي أنشأناه اسما , وهو Obj , ومن خلال هذا الاسم يمكننا التعامل مع الكائن .

عودة إلى الإجراء الخاص بتحميل الفورم , والذي كتبنا فيه بعض السطور , بعد هذه السطور نكتب , أمر إنشاء الكائن كالتالي :

```
Obj.Create Dd
```

لاحظ أن الأمر Create , ما هو إلا وظيفة أنشأناها بأنفسنا في أول برمجتنا للكائن , وأما Dd , فهو يشير إلى Direct Draw.

الآن نبدأ الحلقة التكرارية :

```
Do

DoEvents


Loop
```

فيما بين DoEvents و Loop , نكتب أوامر الرسم , والتحريك كالتالي :

```
Back_Buffer.DrawBox 0, 0, 800, 600

Obj.Move

Obj.Draw

Primary.Flip Nothing, DDFLIP_WAIT
```

لو قمت بتشغيل البرنامج الآن , فسيعمل نظهرا لك شاشة سوداء , وفي أعلى يمينها صورة الكائن الذي أنشأناه , ولكن بدون أي حركة ...

اذهب إلى إجراء الضغط على زر , KeyDown , من إجراءات الفورم , وفيه أكتب التالي :

```
If KeyCode = vbKeyRight Then Obj.Speedx = 6
If KeyCode = vbKeyLeft Then Obj.Speedx = -6
If KeyCode = vbKeyUp Then Obj.Speedy = -6
If KeyCode = vbKeyDown Then Obj.Speedy = 6
```

الآن قم بتشغيل البرنامج , ثم استخدم الأسهم للتحريك , وانظر ماذا ترى :

تجد أن الكائن يتحرك تبعا للسهم الذي تضغط عليه , ولكن بدون توقف , أي أن الكائن يظل متحركا في اتجاهه حتى نهاية الشاشة , وهذا يدفعنا إلى الإجراء KeyUp من اجراءات الفورم , وفيه نكتب التالي :

```
If KeyCode = vbKeyRight Then Obj.Speedx = 0
If KeyCode = vbKeyLeft Then Obj.Speedx = 0
If KeyCode = vbKeyUp Then Obj.Speedy = 0
If KeyCode = vbKeyDown Then Obj.Speedy = 0
```

لاحظ أننا :

١-عرفنا الكائن

٢-جعلنا اسم الكائن Obj

٣- وضعنا فيه الصفات والوظائف

٤-نقوم بتغيير الصفات عن طريق الأزرار

٥-نقوم باستخدام الوظائف من خلال الحلقة التكرارية Loop

كان هذا مثالا مبسطا عن برمجة الكائنات من خلال Direct Draw ولأنه مجرد مثال تعليمي فهو قد لا يوضح لك أهمية البرمجة باستخدام الكائنات , ولكنك مع التجربة , وإنشاء كائناتك الخاصة بنفسك , فستتعرف تدريجيا على أهمية برمجة الكائنات .

وقد خصصت التطبيق الثالث من مجلد التطبيقات , كمثال على عمل CLASS MODULE .

Direct Draw 7

Window Mode

لا شك أن جميع الألعاب , والبرامج , أو معظمها , التي تستخدم Direct Draw تعمل في نظام كامل الشاشة Full-Screen , ولكن لما كان هذا كتاب لشرح Direct Draw فلابد من تضمين بعض الشرح , لكيفية عمل Direct Draw في نافذة , أو Window Mode .

قم بعمل فورم جديدة , وغير قيمة Border Style لها إلى 1-Fixed Single , ثم غير قيم الارتفاع , والعرض كما تحب ...

قبل أن أبدأ في وضع الكود , يجب أن نتعرف على بعض الأشياء ...

تقاس الشاشة في نظام DirectX بالنقط أو Pixels , في حين تقاس الفورم العادية بوحدات قياس أصغر تسمى Twips , ولما كنا سنستخدم Direct Draw من خلال الفورم , فيجب علينا أن نوحد وحدة القياس بينهما , و دائما , على معظم الأجهزة , يمثل الفارق بين Pixels و Twips بالمعادلة التالية :

Pixel = 15 Twip

ولكن لما كان هذا غير ثابت على كل الأجهزة , فقد وجب أن تجعل برنامجك يحول من Twip إلى Pixel على الجهاز الذي يعمل عليه , مهما كان الفارق بين Pixel و Twip على هذا الجهاز .

وعلى هذا فإن عرض الفورمة بوحدة Twip يمثل كالتالي :

```
Width = Form1.Width
```

ويمثل بوحدة Pixels كالتالي :

```
Width = Form1.Width / Screen.TwipPerPixelX
```

وارتفاعها بوحدة Pixel كالتالي :

```
Height = Form1.Height / Screen.TwipPerPixelY
```

وهكذا , نكون قد وحدنا بين وحدتي القياس بين النافذة و Direct Draw .

عودة إلى البرنامج , ونبدأ بوضع المتغيرات الأساسية كالتالي :

```
Dim Dx As New DirectX7
Dim Dd As DirectDraw7
Dim Primary As DirectDrawSurface7
Dim Surf As DirectDrawSurface7
Dim Clipper As DirectDrawClipper
Dim Ddsd As DDSURFACEDESC2
Dim R1 As RECT, R2 As RECT
```

الجديد هنا أننا وضعنا متغيرا جديدا وهو من نوع DirectDrawClipper ,
وهو متغير ضروري للعمل بنظام Window-Mode .

وأننا لم نضع سطح خلفي Back_Buffer , وذلك لأننا لا نحتاج في نظام
النافذة إلى سطح خلفي , بل نقوم بالرسم على الرئيسي دفعة واحدة .

نبدأ بتعيين Direct Draw كالتالي :

```
Set Dd = Dx.DirectDrawCreate("")
```

ثم مستوى التعامل مع كارت العرض كالتالي :

```
Dd.SetCooperativeLevel Me.hWnd, DDSCL_NORMAL
```

لاحظ أننا لا يمكننا أن نغير نظام العرض Display Mode أثناء التعامل مع نافذة , ولكن يمكن فقط لو كان العمل بحجم الشاشة .

بعد هذا نستخدم Ddsd في تحديد صفات السطح الرئيسي كالتالي :

```
Ddsd.lFlags = DDSD_CAPS
Ddsd.ddsCaps.lCaps = DDSCAPS_PRIMARYSURFACE
```

ثم ننشئ السطح الرئيسي كالتالي :

```
Set Primary = Dd.CreateSurface(Ddsd)
```

ثم نبدأ في إعداد Ddsd لنضع فيه صفات السطح الذي سيحمل الصورة كالتالي :

```
Ddsd.ddsCaps.lCaps = DDSCAPS_OFFSCREENPLAIN
Ddsd.lHeight = Me.Height / Screen.TwipsPerPixelY
Ddsd.lWidth = Me.Width / Screen.TwipsPerPixelX
```

ثم ننشئ السطح الذي سيحمل الصورة كالتالي :

```
Set Surf = Dd.CreateSurfaceFromFile(App.Path &
"\pic.bmp", Ddsd)
```

بعد هذا ننشئ الكائن Clipper , ونحدد مقبض النافذة له كالتالي :

```
Set Clipper = Dd.CreateClipper(0)
Clipper.SetHWnd Me.hWnd
```

ثم نضع أبعاد الفورم في Rect كالتالي :

```
Dx.GetWindowRect Me.hWnd, R2
```

ثم نعبر عن أبعاد الـ Rect الآخر كالتالي :

```
With R1
  .Top = 0
  .Left = 0
```

```
  .Right = Me.Width / Screen.TwipsPerPixelX
  .Bottom = Me.Height / Screen.TwipsPerPixelY
End With
```

فأما Rect الأول فيحمل أبعاد النافذة , وأما Rect الثاني فيحمل أبعاد السطح الذي سيتم رسمه .

الآن نبدأ الحلقة التكرارية :

```
Do
DoEvents
Primary.Blt R2, Surf, R1, DDBLT_WAIT
Loop
```

هذا هو كل شيء , وهو سهل جدا كما ترى , وقد خصصت التطبيق الرابع من مجلد التطبيقات كمثال على عمل Direct Draw Window Mode .

هذا هو كل شيء بالنسبة لـ Direct Draw , وأرجو ألا أكون قد نسيت شيئا , كما أتمنى أن يكون كل ما سبق مفهوما تماما .

الأصوات

Direct Sound

نستخدم نفس الطريقة التي استخدمناها في كارت العرض مع Direct Draw في التعامل مع كارت الصوت في Direct Sound من حيث تحديد عدد السواقات المتوفرة , وتعيين أسماءها , ورموزها ...

تعلن عن كائن تحديد عدد الكروت كالتالي :

```
Dim DsE As DirectSoundEnum
```

ثم نعلن عن متغيرات نستخدمها في البرنامج كالتالي :

```
Dim dCount As Long
Dim dNames() As String
Dim dGUIDs() As String
Dim I As Long
```

ثم ننشئ الكائن DsE باستخدام Dx كالتالي :

```
Set DsE = Dx.GetDSEnum
```

بعد هذا نضع قيمة عدد السواقات المتوفرة في المتغير dCount كالتالي :

```
dCount = DsE.GetCount
```

ثم نعيد تعيين قيم الأسماء , والرموز :

```
ReDim dNames(dCount)
ReDim dGUIDs(dCount)
```

والآن , عن طريق حلقة تكرارية من نوع For...Next نضع قيم الأسماء والرموز في المتغيرات التابعة لها :

```
For I = 1 To dCount
dNames(I) = DsE.GetDescription(I)
dGUIDs(I) = DsE.GetGuid(I)
Next
```

وهكذا , حيث أنك في مجلد الأمثلة , في المثال التاسع , ستجد مثالا يطبق ذلك , على Direct Sound.

لكي نبدأ في تحميا صوت من ملف موجي من نوع *.Wav يجب أن نستعين بكائنين , الأول هو DSBUFFERDESC و WAVEFORMATEX , وأول شيء هو اعداد كائن Direct Sound نفسه , ويتم الإعلان عنه كالتالي :

```
Dim Ds As DirectSound
```

ثم يتم انشاؤه باستخدام كائن DirectX كالتالي :

```
Set Ds = Dx.DirectSoundCreate("")
```

ولاحظ أن القوسين " " يعنيان أننا سنستخدم سواقة كارت الصوت الرئيسي , ويمكننا اختيار أي كارت آخر إذا عرفنا GUID الخاص به , ثم تحديد مستوى التحكم في كارت الصوت كالتالي :

```
Ds.SetCooperativeLevel Me.hWnd, DSSCL_PRIORITY
```

بعد هذا يتم الإعلان عن كائن من نوع DSBUFFERDESC كالتالي :

```
Dim Dsbd As DSBUFFERDESC
```

وفيه نضع خصائص الصوت كالتالي :

```
Dsbd.lFlags = DSBCAPS_CTRLVOLUME Or _
DSBCAPS_CTRLFREQUENCY Or DSBCAPS_CTRLPAN
```

ثم نعلن عن كائني السطح الصوتي , وخصائص الصوت كالتالي :

```
Dim sBuffer As DirectSoundBuffer
Dim Wa As WAVEFORMATEX
```

وبعد هذا نقوم بتحميل الصوت في السطح عن طريق كائن Direct Sound كالتالي :

```
Set sBuffer = Ds.CreateSoundBufferFromFile(App.Path _
& "\w.wav", Dsbd, Wa)
```

الآن أصبح الصوت جاهزا للتشغيل , حيث يمكن تشغيله بحيث يعيد التشغيل كلما انتهى الصوت كالتالي :

```
sBuffer.Play DSBPLAY_LOOPING
```

كما يمكن تشغيله بحيث يعمل مرة واحدة فقط , ولا يعيد التشغيل كالتالي :

```
sBuffer.Play DSBPLAY_DEFAULT
```

وقد وضعت المثال العاشر من الأمثلة , في مجلد الأمثلة , لكيفية تحميل صوت من ملف Wave على سطح صوتي , ثم تشغيله .

التأثيرات على الأصوات

يوفر لك Direct Sound العديد من التأثيرات على الأصوات , مثل التحكم في ارتفاع الصوت Volume , والتحكم في التردد Frequency , كذلك يتيح لك معرفة طول الملف الصوتي , والموقع الحالي للمؤشر أثناء تشغيل الصوت .

اصنع كما في المثال السابق , وحمل صوتا ما في السطح الصوتي الذي باسم dsBuffer , وبعد ذلك يمكنك معرفة ارتفاع الصوت من خلال :

```
DsBuffer.GetVolume
```

والقيمة التلقائية له هي ٠ , وهو أعلى صوت يمكن تشغيله من خلال Direct Sound وكلما أردت تخفيض الصوت تقلل القيمة بحيث تكون –٢٠٠ أو أعلى أو أقل , وذلك عن طريق :

```
DsBuffer.SetVolume –200
```

كما يتيح لك معرفة تردد الملف كما بالتالي :

```
DsBuffer.GetFrequency
```

ويمكنك تغيير هذا التردد الذي سيكون له تأثيرا كبيرا على الصوت الناتج على الملف الصوتي , ويتم تغيير التردد كما بالتالي :

```
DsBuffer.SetFrequency 20000
```

والملاحظ هنا أن Frequency تأخذ قيما كبيرة كـ ١٥٠٠ أو ١٠٠٠٠ وهكذا , كما يمكنك معرفة طول الملف الصوتي عن طريق الكائن DSBUFFERDESC كالتالي :

```
Dsbd.lBufferBytes
```

ويمكننا أن نعرف الموضع الحالي للسطح أثناء تشغيل الصوت عن طريق متغير من نوع DSCURSORS الذي يتم الإعلان عنه كالتالي :

```
Dim Cur As DSCURSORS
```

ثم يتم وضع الموضع الحالي للصوت كالتالي :

```
GetCurrentPosition Cur
```

ولمعرفة الموضع الحالي للصوت نستخدم Cur كالتالي :

```
Cur.lPlay
```

تسجيل الأصوات

Sound Capture

من أهم قدرات DirectX في التعامل مع الأصوات , هي قدرتها على تسجيل الأصوات من ميكروفون Microphone , وتسجيلها على القرص الصلب , أو على سطح صوتي مؤقت , وهذا ليس صعبا جدا في الواقع , ولكنه موضوع غير مشهور , على الرغم من أهميته .

عملية تسجيل الأصوات تحتاج لأربعة إجراءات , الأول هو إلتقاط الصوت , من ميكروفون , والثاني هو تحويل الصوت إلى سطح صوتي , والثالث هو السماح للمستخدم بسماع الصوت قبل تخزينه على القرص الصلب , والأخير هو تخزين الصوت على القرص الصلب .

دعنا نبدأ العمل , أنشئ مشروعا جديدا , وضع في الفورم الرئيسية أربعة أزرار أوامر , غير عناوينها إلى التالي :

١-ابدأ التسجيل

٢-أوقف التسجيل

٣-سماع الصوت

٤-حفظ إلى ملف

ثم اذهب إلى صفحة الكود , وأعلن عن المتغيرات الرئيسية كالتالي :

```
Dim Dx As New DirectX7

Dim DSCapture As DirectSoundCapture

Dim DSCaptureBuffer As DirectSoundCaptureBuffer

Dim DSCDesc As DSCBUFFERDESC


Dim DS As DirectSound

Dim DSBuffer As DirectSoundBuffer

Dim DSDesc As DSBUFFERDESC

Dim Cur As DSCURSORS

Dim Wa As WAVEFORMATEX

Dim BufferBytes() As Integer
```

أما DSCapture فهو العنصر الرئيسي في تسجيل الأصوات , و

DSCaptureBuffer فهو السطح الذي سنسجل الصوت عليه , وأما

DSCDesc فستحمل صفات هذا السطح .

نبدأ في إعداد المتغيرات , في إجراء تحميل الفورم كالتالي :

```
Set DSCapture = Dx.DirectSoundCaptureCreate("")

Set DS = Dx.DirectSoundCreate("")

DS.SetCooperativeLevel Me.hWnd, DSSCL_NORMAL
```

ثم نبدأ في تحديد صفات الملف الصوتي كالتالي :

```
With Wa
 .nFormatTag = WAVE_FORMAT_PCM
 .nChannels = 2
 .lSamplesPerSec = 22050
 .nBitsPerSample = 16
 .nBlockAlign = 4
 .lAvgBytesPerSec = 88200
 .nSize = 0
End With
```

ثم نحدد قيم متغيرات العنصر DSCDesc كالتالي :

```
DSCDesc.fxFormat = Wa

DSCDesc.lBufferBytes = Wa.lAvgBytesPerSec * 20
```

```
DSCDesc.lFlags = DSCBCAPS_WAVEMAPPED
```

ثم – بناء على ما سبق – نبدأ في إنشاء السطح الذي سيتم التسجيل عليه كالتالي :

```
Set DSCaptureBuffer =

DSCapture.CreateCaptureBuffer(DSCDesc)
```

الآن نذهب إلى زر (ابدأ التسجيل) , وفي كوده نكتب :

```
DSCaptureBuffer.start DSCBSTART_DEFAULT
```

ثم إلى زر (إيقاف التسجيل) وفيه نكتب :

```
DSCaptureBuffer.Stop
```

والآن إلى زر (شغل الصوت) وفيه نكتب :

قبل أن نكتب الكود يجب أن تعرف الخطوات , أول خطوة هي معرفة معلومات عن الصوت الذي تم تسجيله , ثم نقل هذا الصوت من سطح تسجيل , إلى سطح صوتي عادي , ثم تشغيل الصوت من هذا السطح العادي , كالتالي :

```
DSCaptureBuffer.GetCurrentPosition Cur
If Cur.lWrite = 0 Then
MsgBox "there is no records to play"
Exit Sub
End If
DSDesc.lBufferBytes = Cur.lWrite *
DSCDesc.fxFormat.nBlockAlign
DSDesc.lFlags = DSBCAPS_CTRLVOLUME Or DSBCAPS_STATIC
Set DSBuffer = DS.CreateSoundBuffer(DSDesc,
DSCDesc.fxFormat)
ReDim BufferBytes(Cur.lWrite *
DSCDesc.fxFormat.nBlockAlign + 1)
DSCaptureBuffer.ReadBuffer 0, Cur.lWrite *
DSCDesc.fxFormat.nBlockAlign, BufferBytes(0),
DSBLOCK_DEFAULT
DSBuffer.WriteBuffer 0, Cur.lWrite *
DSCDesc.fxFormat.nBlockAlign, BufferBytes(0),
DSBLOCK_DEFAULT
DSBuffer.Play DSBPLAY_DEFAULT
```

وفي زر(حفظ إلى ملف) , نكتب :

```
DSBuffer.SaveToFile App.Path & "\WaveFile.wav"
```

هذا كل شيء , وقد وضعت التطبيق الخامس , من مجلد التطبيقات , كمثال لبرنامج يلتقط الأصوات , ويخزنها على القرص الصلب , مع إمكانية تشغيلها قبل تخزينها .

هنا ننتهي من Direct Sound , ثم نبدأ في تطبيق قريب جدا منه وهو Direct Music .

دايركت ميوزيك

Direct Music

عرفنا من قبل أن الأصوات , والمؤثرات الصوتية , لا غنى عنهما في أي برنامج وسائط متعددة , أو لعبة , ولذا فقد درسنا معا Direct Sound , ولكن Direct Sound لا تعمل إلا مع الملفات من نوع Wav فقط , أو الملفات الموجية Wave Files كما يقال عنها .

هناك نوع آخر مهم من الأصوات , لا يدعمه دايركت ساوند , وهو ملفات الموسيقى Music Files , والتي تأخذ دائما الامتداد *.mid , وهو نوع مهم حقا من الملفات , وسبب أهميته يعود إلى صغر حجمه الكبير عن الملفات الموجية , حيث أنك يمكن أن تجد ملف موسيقى يستغرق أكثر من ثلاثة دقائق , ويكون حجمه أقل من ٥٠ KB .

وتسمى ملفات MIDI بملفات الموسيقى لأنها لا يمكن أن تخرج سوى اصوات الآلات الموسيقية , نعرف أن Wave يمكنه اخراج أي صوت , مثل انفجار , انزلاق , أو حتى صوت الأشخاص , والحيوانات , أما Midi فلا تخرج سوى اصوات الآلات الموسيقية , مثل البيانو , والكمان , والجيتار , وغير ذلك .

تعال نبدأ العمل , أنشئ مشروعا جديدا , وأدخل فيه مرجع Reference الخاص بـ DirectX7 ثم اذهب إلى صفحة الكود الخاص بالفورم , وفيه أعلن عن المتغيرات الرئيسية , كالتالي :

```
Dim Dx As New DirectX7
Dim DmLoad As DirectMusicLoader
Dim DmPer As DirectMusicPerformance
Dim Seg As DirectMusicSegment
```

والمتغير الأخير Seg هو الذي ستم تحميل الملف الموسيقي عليه , كالسطح في Direct Draw و الـ Buffer في Direct Sound.

وفي إجراء تحميل الفورم , نكتب :

```
Set DmLoad = Dx.DirectMusicLoaderCreate
Set DmPer = Dx.DirectMusicPerformanceCreate
```

هذا لانشاء المتغيرات الرئيسية :

```
DmPer.Init Nothing, Me.hWnd
```

وهذا لتنصيب DmPer , ونحن نضع Nothing إذا لم نكن نستخدم Direct Sound في البرنامج , أما لو كنا نستخدم Direct Sound فإننا نضع رمز Direct Sound بدلا من Nothing , فيكون كالتالي :

```
DmPer.Init Ds, Me.hWnd
```

ثم نقوم بتحديد الـ Port كالتالي :

```
DmPer.SetPort -1, 1
```

ثم نقوم بتحميل الملف الصوتي في المتغير Seg كالتالي :

```
Set Seg = DmLoad.LoadSegment(App.Path & _
"\midi.mid")
```

بعد هذا نضع بعض التعريفات , التي لا غنى عنها كالتالي :

```
Seg.SetStandardMidiFile
DmPer.SetMasterAutoDownload True
```

```
Seg.Download DmPer
```

هكذا تم تحميل الملف , وهو جاهز للتشغيل , ولتشغيله نكتب :

```
DmPer.PlaySegment Seg, 0, 0
```

ولإيقافه نكتب :

```
DmPer.Stop Seg, Nothing, 0, 0
```

وقد وضعت مثالا على تشغيل Direct Music في مجلد DM_SAMPLES , في مجلد فرعي باسم Sample11 .

التأثيرات في Direct Music

تقدم لك Direct Music عددا من التأثيرات التي يمكنك استخدامها في تعاملك مع الموسيقى , ولكن يجب أولا التعرف على كائن جديد وهو DirectMusicSegmentState , وهو عبارة عن كائن يستخدم في تشغيل , واختبار , وإيقاف Segment أو Seg كما نعرفه نحن .

أولا ننشئ كائن من النوع DirectMusicSegmentState كالتالي :

```
Dim State As DirectMusicSegmentState
```

ثم نتعامل مع برنامجنا بطريقة طبيعية كالتالي :

```
Set DmLoad = Dx.DirectMusicLoaderCreate
Set DmPer = Dx.DirectMusicPerformanceCreate
DmPer.Init Nothing, Me.hWnd
DmPer.SetPort -1, 1
Set Seg = DmLoad.LoadSegment(App.Path & _
"\midi.mid")
Seg.SetStandardMidiFile
DmPer.SetMasterAutoDownload True
Seg.Download DmPer
```

ولكننا سنغير كود بدء التشغيل ليكون هكذا :

```
Set State = DmPer.PlaySegment(Seg, 0, 0)
```

وكود الإيقاف ليكون هكذا :

```
DmPer.Stop Seg, State, 0, 0
```

التأثير الأول , تسريع التشغيل :

```
DmPer.SetMasterTempo 2
```

والرقم ٢ يعني أن التشغيل سيكون بضعف سرعته العادية .

التأثير الثاني , تشغيل جزء فقط من الملف الموسيقي , وليس الملف كله :

```
Seg.SetStartPoint 500
Seg.SetLength 20000
```

التأثير الثالث , معرفة موقع الجزء الحالي , أثناء التشغيل :

```
State.GetSeek
```

التأثير الرابع , عمل Looping للملف الموسيقي :

```
Do
DoEvents
```

```
If State.GetSeek = Seg.GetLength Then

Seg.SetStartPoint 500

Set State = Nothing

Set State = DmPer.PlaySegment(Seg, 0, 0)

End If

Loop
```

هذا كل شيء بالنسبة لـ Direct Music , وقد خصصت المثال في المجلد DM_SAMPLES في المجلد Sample12 باسم لموضوع التأثيرات على Direct Music.

التعامل مع وسائل الإدخال باستخدام

Direct Input

تعاملنا في دروسنا السابقة , في درس برمجة الكائنات , كيفية التحريك باستخدام لوحة المفاتيح Key Board , وقد استخدمنا في ذلك الإجراء Key Down من اجراءات Form , وهناك أيضا الإجراء Key Press , ولكن كلتا الإجرائين , كما رأينا في التجربة , لا تصلحان للتعامل مع الألعاب الكبيرة ...

أو , بمعنى أقرب للدقة , هما لا تصلحان للتعامل مع الألعاب بتاتا , وهناك أيضا دالة من دوال API يمكن استخدامها في التعامل مع لوحة المفاتيح , ولكنها أيضا وسيلة ناقصة , ولا تبلغ أبدا إمكانيات Direct Input .

تمكنك Direct Input بكفاءة من التعامل مع كافة وسائل الإدخال , سواء كانت لوحة مفاتيح , أو الفأرة , أو حتى عصا الألعاب , ونحن في هذا الكتاب سنتعامل مع Direct Input مع لوحة المفاتيح فقط , أما الماوس فسأضع بعض الشروح للتعامل معه باستخدام دوال API .

لنبدأ العمل , بعد إنشاء مشروعا جديدا نضع في كود الفورم له الإعلانات الرئيسية كالتالي :

```
Dim Dx As New DirectX7
Dim Di As DirectInput
Dim Didev As DirectInputDevice
Dim DiKey As DIKEYBOARDSTATE
```

الكائن Di هو كائن Direct Input الرئيسي , وكائن Didev هو كائن التحكم في سواقة الإدخال , وهي في مثالنا KeyBoard , وأما DiKey فهو عبارة عن Type يحتوي على جميع أزرار لوحة المفاتيح , وهو الذي سنستقبل إشارات لوحة المفاتيح من خلاله .

وأما إنشاء كائن Direct Input فيكون كالتالي :

```
Set Di = Dx.DirectInputCreate
```

لاحظ هنا اننا لم نضع أي Guid في تعريف Direct Input على عكس Direct draw و Direct Sound وذلك لأننا نضع Guid حسب نوع السواقة التي سنستخدمها من خلال Didev , وهذا يتم كالتالي :

```
Set Didev = Di.CreateDevice("guid_syskeyboard")
```

ولمتابعة تعيين خصائص Didev :

```
Didev.SetCooperativeLevel Me.hWnd, DISCL_NONEXCLUSIVE
Or DISCL_BACKGROUND
Didev.SetCommonDataFormat DIFORMAT_KEYBOARD
```

وللبدء في الاتصال بلوحة المفاتيح نكتب :

```
Didev.Acquire
```

هكذا نكون مستعدين للاتصال بلوحة المفاتيح , ويفضل إنشاء وظيفة Function خاصة للتعامل مع لوحة المفاتيح , ولتكن كالتالي :

```
Function GetKeys()

End Function
```

وأول ما نضع في هذه الوظيفة هي أمر كائن Didev بمعرفة إشارات لوحة المفاتيح , وتخزينها في كائن لوحة المفاتيح Dikey كالتالي :

```
Didev.GetDeviceStateKeyboard DiKey
```

بعد هذا نتعامل مع DiKey من خلال جملة If كالتالي :

```
If DiKey.Key(DIK_UP) Then MoveUp زر الأعلى
If DiKey.Key(DIK_DOWN) Then MoveDown زر الأسفل
If DiKey.Key(DIK_LEFT) Then MoveLeft زر السهم لليسار
If DiKey.Key(DIK_RIGHT) Then MoveRight زر السهم لليمين
If DiKey.Key(DIK_ESCAPE) Then End زر الهروب
```

نعود إلى الكود في حدث تحميل الفورم , وفيه نكتب :

```
Do
DoEvents
GetKeys
Loop
```

وهكذا , وقد خصصت المثال الحادي عشر , في مجلد الأمثلة , لذلك حيث وضعت فيه List بكل الأزرار المحتملة , كما يمكنك تشغيل المثال , ثم الضغط على أي زر , حيث سيتجه مؤشر List إليه مباشرة , عدا بعض الأزرار التي لا تقبلها List , كالاتجاهات الأربعة .

التعامل مع الفأرة

على الرغم من أنني لن أشرح التعامل مع الفأرة , من خلال DirectX , وأن هذا الكتاب مخصص فقط لـ DirectX , إلا أنني سأضع بعد طرق التعامل مع الفأرة من خلال Visual Basic .

أول شيء في التعامل مع الفأرة هو معرفة الزر المضغوط , فكما نعرف أن الفأرة يكون بها زرين , أو ثلاثة , وطريقة معرفة الزر المضغوط هامة جدا , لو كان لكل زر وظيفة , تختلف عن الزر الآخر .

يتم معرفة الزر الذي تم ضغطه من الـ Mouse عن طريق المتغير Button , هذا ضمن الإجراء Mouse Down , للأداة المراد اختبار الضغط عليها .

لو كانت قيمة Button تساوي ١ فإن الزر الذي تم الضغط عليه هو زر الفارة الأيسر , وأما لو كانت قيمة Button تساوي ٢ , فإن الزر الأيمن هو الذي تم الضغط عليه , وأما لو كانت ٤ فإن الزر الأوسط هو الذي تم الضغط عليه , هذا مثلا مثال ينشئ صندوق رسالة , إذا ضغط المستخدم زر الفأرة الأيمن , على الفورم

```
Private Sub Form_MouseDown(Button As Integer, Shift
As Integer, X As Single, Y As Single)
```

```
If Button = 2 Then MsgBox "Right Click"
End Sub
```

وكما هو موضح من الكود السابق , يوضع كود الكشف على ضغطة الفأرة , في الإجراء MouseDown .

نحن نعرف بالطبع أنه يمكن استخدام الفأرة في اختيار , وتحديد أحد العناصر , كما يمكن استخدام الزر Ctrl و الزر Shift لاختيار أكثر من عنصر , فكيف يتم معرفة إذا كان المستخدم يضغط على أحد هذين الزرين , أو كلاهما , أثناء ضغطه على زر الفأرة ؟؟

يتم ذلك من خلال معرفة قيمة Shift , والتي تأخذ قيمة من ثلاثة :

١ = تم ضغط زر Shift مع الفأرة .

٢ = تم ضغط زر Ctrl مع الفأرة .

٣ = تم ضغط زر Shift و Ctrl مع الفأرة .

ويتم اكتشاف قيمة Shift أيضا من خلال الإجراء MouseDown للأداة , كالتالي :

```
If Shift = 1 Then MsgBox "Shift Key"
```

قد تحتاج في بعض الأحيان إلى اكتشاف موقع المؤشر على الفورم , أو على أي أداة , وفي ذلك يمكن استخدام MouseDown أيضا , ولكن يفضل استخدام MouseMove , ويكون الإكتشاف عن طريق المتغيرين X , و Y , كالتالي :

```
Caption = X & " * " & Y
```

والملاحظ هنا أن X و Y تعملان فقط إذا كان المؤشر داخل حدود الفورم , أما لو خرج عن حدودها , فإنها تتوقف عن الحساب , والملاحظ أيضا أنها تعتبر أن أعلى يسار الفورم هو النقطة 0 و 0 , وليس أعلى يسار الشاشة .

كذلك يمكن اكتشاف موقع المؤشر , باستخدام دوال API كالتالي :

أو لا إضافة دالة API, إلى قسم الإعلان العام كالتالي :

```
Private Declare Function GetCursorPos Lib "user32"
(lpPoint As POINTAPI) As Long
```

بعد هذا نعلن عن Type من نوع POINTAPI كالتالي :

```
Private Type POINTAPI

x As Long

y As Long

End Type
```

ثم نعلن عن متغير من نوع هذا الـ Type كالتالي :

```
Dim P As POINTAPI
```

ثم نذهب إلى اجراء Form_Load ونكتب كود جلب موضع المؤشر , وطباعته في مكان عنوان الفورم كالتالي :

```
Show

Do

DoEvents

GetCursorPos P

Caption = P.x & " * " & P.y

Loop
```

ونلاحظ هنا أن الدالة GetCursorPos تستخدم Pixels في قياساتها , كما أنها تعتبر أن نقطة الصفر هي أعلى يسار الشاشة , وليس الفورم .

وهناك دالة أخرى تساعدنا على وضع المؤشر أينما نشاء , وهي الدالة :

```
Private Declare Function SetCursorPos Lib "user32"
(ByVal x As Long, ByVal y As Long) As Long
```

ثم في اللحظة التي نريد فيها تغيير موضع المؤشر نستخدم :

```
SetCursorPos 20, 100
```

ويمكنك أن تضع أي رقمين بدلا من ٢٠ , و ١٠٠ .

ويمكننا تحديد مكان وجود الوؤشر داخل الفورم باستخدام دالتين هما :

```
Private Declare Function GetWindowRect Lib "user32"
(ByVal hwnd As Long, lpRect As RECT) As Long
```

هذه الدالة لمعرفة مساحة , ومكان النافذة , والدال الأخرى :

```
Private Declare Function ClipCursor Lib "user32"
(lpRect As Any) As Long
```

وتستخدم في تحديد مكان المؤشر , والدالتان تستخدمان Rect , وهو نفس النوع الذي نستخدمه في DirectX , بناؤه كالتالي :

```
Private Type RECT
        Left As Long
        Top As Long
        Right As Long
        Bottom As Long
End Type
```

أول شيء هو وضع متغيرين من نوع Rect , الأول سيحمل مقاييس نظام العرض عندك , والآخر سيحمل مساحة النافذة , كالتالي :

```
Dim FullScreenRect As RECT
Dim WindowRect As RECT
```

وفي إجراء تحميل الفورم , نبدأ في تحديد قيم الـ RECT الأول , والذي سيحمل مقاييس نظام العرض , كالتالي :

```
With FullScreenRect
      .Top = 0
      .Left = 0
      .Right = Screen.Width / Screen.TwipsPerPixelX
      .Bottom = Screen.Height / Screen.TwipsPerPixelY
End With
```

بعد هذا نعين خصائص الـ Rect الآخر بواسطة الدالة GetWindowRect التي ستضع فيه قيم إحداثيات , ومساحة الفورم كالتالي :

```
GetWindowRect Me.hwnd, WindowRect
```

ثم نحدد مكان المؤشر ليظل في داخل الفورم , باستخدام Rect النافذة كالتالي :

```
ClipCursor WindowRect
```

هكذا لن يتحرك المؤشر أبدا خارج إحداثيات الفورم , ولكنه سيظل محبوسا في هذه المنطقة حتى لو أغلقت الفورم , والحل هو تحريره قبل إغلاق الفورم , ويتم ذلك في إجراء Form_Unload , كالتالي :

```
ClipCursor FullScreenRect
```

وقد خصصت المثال الثاني عشر , من مجلد الأمثلة , ليكون مثالا عن استخدام الدالتين الأخيرتين .

هذا كل شيء عن التحكم , والتعامل مع الفأرة Mouse , وكذلك كل شيء بالنسبة للتعامل مع Direct Input .

التطبيقات ثلاثية الأبعاد باستخدام

Direct3D Immediate Mode

لا شك أن قوة DirectX تكمن في قدرتها على تطوير , وبناء البرامج , ذات الأبعاد الثلاثة , والتي أصبحت منتشرة جدا الآن , وكذلك لا شك أن DirectX توفر لك إمكانيات مذهلة في هذا المجال .

يوفر لك Direct3D لإمكانيات هائلة , وغير محدودة , في التعامل مع العوالم ثلاثية الأبعاد 3D Worlds , من حيث الإنشاء , والتعديل , والإكساء , والإضاءة , وتغيير الأحجام , والتحريك , وغير ذلك كثيرا من التأثيرات .

سنبدأ معا بمثال بسيط , يوضح لنا كيفية رسم مضلع صغير , مربع , متعدد الألوان , التي تتلاقى مع بعضها بتدرج fading رائع , تعال معا نخترق عالم Direct3D .

أنشئ مشروعا جديدا , ثم أعلن عن المتغيرات التالي :

```
Dim Dx As New DirectX7
Dim Dd As DirectDraw7
Dim Primary As DirectDrawSurface7
```

```
Dim Back_Buffer As DirectDrawSurface7

Dim Ddsd As DDSURFACEDESC2
```

هذه هي متغيرات DirectX المعتادة , والتي تعاملنا معها من قبل , والآن نبدأ مع متغيرات Direct3D :

```
Dim D3d As Direct3D7
```

هذا هو متغير Direct3D العام .

```
Dim Device As Direct3DDevice7
```

هذا هو متغير السواقة ثلاثية الأبعاد .

```
Dim Vertex(3) As D3DTLVERTEX
```

هذه هي النقاط الأربع , المكونة لرؤوس المربع .

```
Dim DENUM As Direct3DenumDevices
```

هذا المتغير سيساعدنا في معرفة Guid للسواقة ثلاثية الأبعاد .

```
Dim D3d_Guid As String
```

هذا المتغير سنحمل فيه قيمة GUID للسواقة ثلاثية الأبعاد .

نبدأ في إعداد كائنات DirectX المعتادة , والتي سيحدث فيها تغيرا طفيفا , كما سترى :

```
Set Dd = Dx.DirectDrawCreate("")
Dd.SetCooperativeLevel Me.hWnd, DDSCL_EXCLUSIVE Or
DDSCL_FULLSCREEN Or DDSCL_ALLOWREBOOT
Dd.SetDisplayMode 800, 600, 16, 0, DDSDM_DEFAULT
Ddsd.lFlags = DDSD_BACKBUFFERCOUNT Or DDSD_CAPS
Ddsd.ddsCaps.lCaps = DDSCAPS_COMPLEX Or DDSCAPS_FLIP
Or DDSCAPS_3DDEVICE Or DDSCAPS_PRIMARYSURFACE
Ddsd.lBackBufferCount = 1
Set Primary = Dd.CreateSurface(Ddsd)
Ddsd.ddsCaps.lCaps = DDSCAPS_BACKBUFFER Or
DDSCAPS_3DDEVICE
```

```
Set Back_Buffer =

Primary.GetAttachedSurface(Ddsd.ddsCaps)
```

والآن ننشئ كائن Direct3D الرئيسي , بمعونة Direct Draw كالتالي :

```
Set D3d = Dd.GetDirect3D
```

بعد هذا نأمر Direct3D بوضع خصائص السواقة ثلاثية الأبعاد , في المتغير المخصص لذلك , كالتالي :

```
Set DENUM = D3d.GetDevicesEnum
```

بعد هذا نضع رمز GUID السواقة , في المتغير الحرفي D3d_Guid كالتالي :

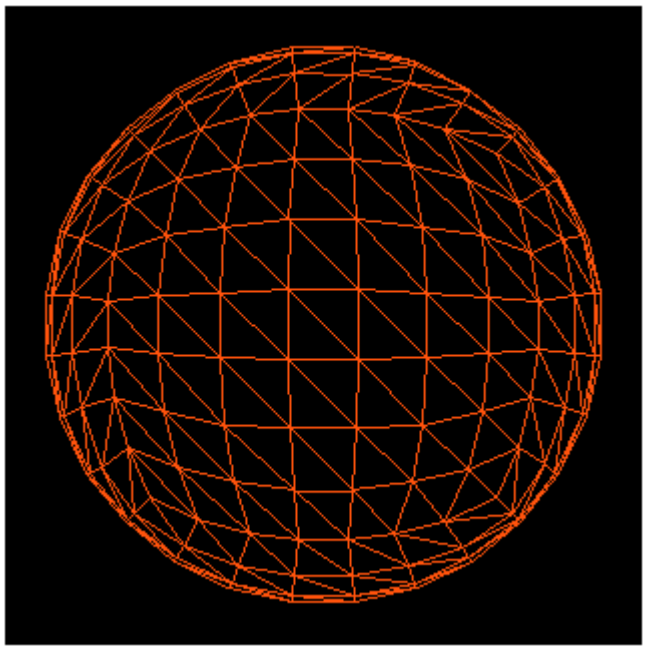
```
D3d_Guid = DENUM.GetGuid(DENUM.GetCount)
```

بعد هذا نقوم بإنشاء السواقة Device , باستخدام GUID و السطح الخلفي , كالتالي :

```
Set Device = D3d.CreateDevice(D3d_Guid, Back_Buffer)
```

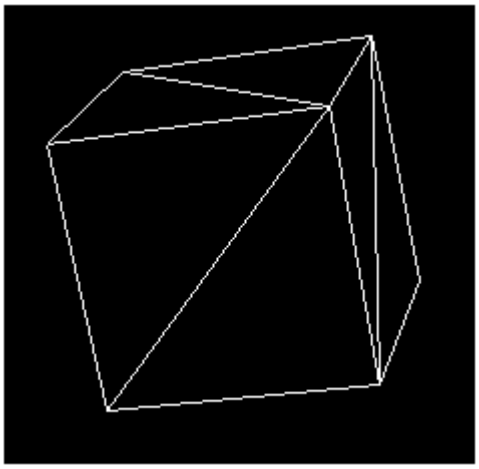
والآن قبل أن نكمل الكود , دعني أعرفك على بعض الأساسيات , في تكوين الشكل باستخدام النقاط .

كل الأشكال المعقدة , تتكون من وحدة بنائية واحدة هي المربعات , أو Rectangles , مثلا الدائرة تكون مكونة من مربعات كالتالي :



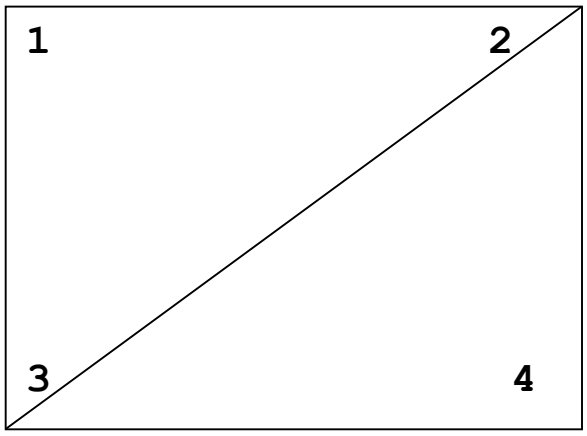
كما ترى , الدائرة مكونة من عدد من المربعات , التي تتجاور على سطحها , مكونة داائرة , أو كرة .

إذن فوحدة بناء المجسمات هي المربع , أو الشكل الرباعي قائم الزوايا , ولكن هذه الوحدة لا تزال غير بسيطة تماما , أو بمعنى أصح , لا تزال هناك وحدات



أبسط منها , وهي المثلث , فالمربع يتكون من مثلثين متلاقيين بالوتر , كما هو مبين بالشكل :

إذن فإن أبسط وحدة بناء , في عالم Direct3D هي المثلث , أو Triangle , وطريقة رسم مربع تتم عن طريق رسم مثلثين متقابلين بالوتر , ونحن سنستخدم طريقة الرسم عن طريق الرؤوس , بطريقة تسمى TRIANGLESTRIP , وهي تعتمد على ترتيب النقاط , فمثلا المربع يجب أن تكون نقاطه مرتبة بالشكل التالي :



مثلا , النقاط التي سنستخدمها لرسم المربع هي كالتالي :

```
 X   -   Y
100   100
300   100
100   300
300   300
```

وعلى هذا يمكننا أن نبدأ في رسم النقاط , ولكن هناك أمر واحد ينبغي أن نتعلمه قبل أن نرسم النقاط , وهو أن DirectX توفر لك إنتاج الألوان بطريق RGB عن طريق الدالة :

```
Dx.CreateColorRGB(r , g , b)
```

والآن لنبدأ في إنشاء النقاط , باستخدام DirectX كالتالي :

```
Dx.CreateD3DTLVertex 100, 100, 0, 1, _
Dx.CreateColorRGB(1, 0, 0), 1, 0, 0, Vertex(0)
```

```
Dx.CreateD3DTLVertex 300, 100, 0, 1, _
```

```
Dx.CreateColorRGB(0, 1, 0), 1, 0, 0, Vertex(1)



Dx.CreateD3DTLVertex 100, 300, 0, 1, _
Dx.CreateColorRGB(0, 0, 1), 1, 0, 0, Vertex(2)



Dx.CreateD3DTLVertex 300, 300, 0, 1, _
Dx.CreateColorRGB(1, 0, 1), 1, 0, 0, Vertex(3)
```

الدالة Dx.CreateD3DTLVertex تأخذ ٩ قيم , الأولى هي القيمة X والثانية هي القيمة Y والثالثة هي قيمة Z ثم قيمة rhw , ثم قيمة Color , ثم قيمة specular , ثم قيمة Tu ثم قيمة Tv , ثم نعطيها رقم النقطة التي ستقوم بإنشائها .

بعد هذا نقوم بإنشاء متغير من نوع D3DRECT , والذي سيقوم بعمل Cls للسواقة كالتالي :

```
Dim ClearRect(0 To 0) As D3DRECT
```

ثم نضع إحداثيات الشاشة فيه كالتالي :

```
ClearRect(0).X2 = 800
ClearRect(0).Y2 = 600
```

ثم نبدأ الحلقة التكرارية :

```
Do
```

نبدأ بعمل CLS للسواقة , باستخدام ClearRect الذي أعلنا عنه كالتالي :

```
Device.Clear 1, ClearRect, D3DCLEAR_TARGET, 0, 0, 0
```

نأمر السواقة بالاستعداد للرسم كالتالي :

```
Device.BeginScene
```

بعد هذا نأمر السواقة برسم المربع / المستطيل الذ أعددنا نقاطه من قبل , كالتالي :

```
Device.DrawPrimitive D3DPT_TRIANGLESTRIP, _
```

```
D3DFVF_TLVERTEX, Vertex(0), 4, D3DDP_WAIT
```

وهي تأخذ عدد من القيم , هي ١- نوع الرسم , ٢- نوع الطريقة التي نستخدمها في الرسم , ٣- أول نقطة نبدأ بها الرسم , ٤- عدد نقاط المجسم , ٥- طريقة الرسم .

بعد هذا نخبر السواقة أننا أنهينا الرسم , كالتالي :

```
Device.EndScene
```

ثم نأمر السطح الرئيسي بعرض الرسومات على الشاشة , كالتالي :

```
Primary.Flip Nothing, DDFLIP_WAIT
```

ثم نغلق الحلقة التكرارية

```
Loop
```

لا تنس أنك ستضع جميع أوامر الرسم بين الأمرين :

```
Device.BeginScene
```

و

```
Device.EndScene
```

وقد صممت المثال الثالث عشر , من مجلد الأمثلة , كمثال على الشرح السابق , في Direct3D .

التعامل مع البعد الثالث

إن أساس روعة , وقوة Direct3D , هي قدرته على تجهيز , والتعامل مع الأبعاد الثلاثة بإتقان شديد , حتى يمكنك من خلاله انشاء عالم ثلاثي الأبعاد يحاكي الطبيعة تماما , بالطبع لن يكون هذا دون أن تكون معك التصميمات اللازمة , ولكني لا أتكلم عن جمال الإنشاء , ولكن عن ديناميكية إنشاء عالم ثلاثي الأبعاد .

سنجهز في هذا الدرس برنامجا عبارة عن مربع (مستطيل) , يرسم على البعد الثالث (Z) , وتكون الكاميرا (الرؤية) مسلطة عليه من أعلى , ويدور حول نفسه .

لعلك بدأت تلاحظ أن الكود يزداد , ويكبر باستمرار , وأنك تضطر لكتابة نفس الكود مرات , ومرات , وهذا في الواقع مفيد , حتى تتسنى لك فرصة التجربة , والممارسة , ولكنه أمر ممل للغاية , لذا فقد بدأت أضع الأوامر العامة , وإنشاء

الكائنات الرئيسية , في وحدة برمجية منفصلة Module , حيث يمكننا , صنعها مرة واحدة فقط , ثم نقوم باستخدامها في أي برنامج , دون الحاجة لكتابتها مرة أخرى .

تعال معا نبدأ العمل , وسنبدأ بتعريف الكائنات العامة Public Variables :

```
Public Dx As New DirectX7
Public Dd As DirectDraw7
Public Primary As DirectDrawSurface7
Public Back_Buffer As DirectDrawSurface7
Public Ddsd As DDSURFACEDESC2

Public D3d As Direct3D7
Public Device As Direct3DDevice7

Dim D3dEnum As Direct3DEnumDevices
Dim Guid As String
```

وقد تعرفنا على كل هذه الكائنات من قبل , وهذه هي الكائنات الجديدة علينا :

```
Public MatWorld As D3DMATRIX

Public MatView As D3DMATRIX

Public MatProj As D3DMATRIX
```

يبدو أن D3DMATRIX هي المتحكمة في كل شيء , بالنسبة للعوالم ثلاثية الأبعاد , سواء من تحريك , أو تدوير , أو حتى عرض على الشاشة , وقد أنشأنا ثلاثة متغيرات من هذا النوع , الأول يتحكم في العالم ثلاثي الأبعاد , والثاني يتحكم في رؤية العنصر (المربع الذي نقوم برسمه) , والثالث يتحكم في موضع , واتجاه الكاميرا .

ثم نعرف ثابتين للقياس بالتقدير الستيني , والتقدير الدائري , حيث أن التقدير الستيني هو تقدير قياس الزاوية الذي نعرفه , وهو يتدرج من صفر إلى ٣٦٠ درجة , أما التقدير الدائري , فهو يساوي الزاوية بالتقدير الستيني , مضروبة في ١٨٠ , مقسومة على 3.14 :

```
Public Const PI As Single = 22 / 7

Public Const Rad As Single = PI / 180
```

حيث أن ٢٢ / ٧ تساوي الرقم الشهير : ٣,١٤

نبدأ في إنشاء كائنات DirectX و Direct3D كالتالي :

```
Set Dd = Dx.DirectDrawCreate("")
Dd.SetCooperativeLevel Form1.hWnd, DDSCL_EXCLUSIVE Or
DDSCL_FULLSCREEN Or DDSCL_ALLOWREBOOT
Dd.SetDisplayMode Width, Height, Bpp, 0,
DDSDM_DEFAULT
Form1.Show
Ddsd.lFlags = DDSD_BACKBUFFERCOUNT Or DDSD_CAPS
Ddsd.ddsCaps.lCaps = DDSCAPS_COMPLEX Or DDSCAPS_FLIP
Or DDSCAPS_3DDEVICE Or DDSCAPS_PRIMARYSURFACE
Ddsd.lBackBufferCount = 1
Set Primary = Dd.CreateSurface(Ddsd)
Ddsd.ddsCaps.lCaps = DDSCAPS_BACKBUFFER Or
DDSCAPS_3DDEVICE
Set Back_Buffer =
Primary.GetAttachedSurface(Ddsd.ddsCaps)


Set D3d = Dd.GetDirect3D
Set D3dEnum = D3d.GetDevicesEnum
Guid = D3dEnum.GetGuid(D3dEnum.GetCount)
```

```
Set Device = D3d.CreateDevice(Guid, Back_Buffer)
```

بعد هذا نتعرف على كائن جديد , وهو D3DVIEWPORT , وهو المسؤول عن حجم الشاشة التي تظهر منها العالم من خلال الكاميرا , ولتقريب الأمر أكثر , نرى في ألعاب سباق السيارات , المرآة , مرآة السيارة , والتي تظهر الطريق من الخلف , وهي في الواقع (بالنسبة للعبة) ليست مرآه , وإنما هي كاميرا تتجه عكس اتجاه السيارة , وتظهر ما تصوره عبر الجزء الممثل للمرآة , وهذا الجزء يتم تحديد أبعاده بواسطة ViewPort , وبما أننا لن نستخدم أكثر من كاميرا واحدة , فإننا سنجعل احداثيات ViewPort بحجم الشاشة بالكامل , كالتالي :

```
Dim ViewPort As D3DVIEWPORT7
With ViewPort
 .lHeight = Height
 .lWidth = Width
 .lX = 0
 .lY = 0
 .maxz = 1
 .minz = 0
End With
```

حيث أن Width تمثل عرض الشاشة , وهي ٨٠٠ في مثالنا , بينما Height تساوي ٦٠٠ , والآن نخبر Device باستخدام ViewPort الذي أنشأناه , كالتالي :

```
Device.SetViewport ViewPort
```

كما قلت من قبل أنني سأضع وحدة برمجية , من أجل تعريف الكائنات الهامة , وكذلك سأضع فيه الوظائف الرئيسية , مثل وظيفة إنتاج D3DVECTOR , ووظيفة إنتاج D3DRECT , فأما الأولى فكالتالي :

```
Function MakeVector(X As Single, Y As Single, Z As
Single) As D3DVECTOR
With MakeVector
  .X = X
  .Y = Y
  .Z = Z
End With
End Function
```

وأما الثانية فكالتالي :

```
Function Make3DRect(X1 As Single, X2 As Single, Y1 As
Single, Y2 As Single) As D3DRECT
With Make3DRect
  .X1 = X1
  .X2 = X2
  .Y1 = Y1
  .Y2 = Y2
End With
End Function
```

حيث توفر علي هاتين الوظيفتين إنشاء متغيرات جديدة من نوع Vector و Rect , وأستخدمها وقت الحاجة فقط .

الآن نعلن عن النقاط الأربع المكونة للمربع , كالتالي :

```
Dim Vertex(3) As D3DLVERTEX
```

لاحظ أننا استخدمنا D3DLVETEX و D3DTLVERTEX كما في المثال السابق , نبدأ في تعيين موضع النقاط الأربع كالتالي :

```
Dx.CreateD3DLVertex 0, 0, 10, _
Dx.CreateColorRGB(1, 1, 1), 1, 0, 0, Vertex(0)

Dx.CreateD3DLVertex 10, 0, 10, _
Dx.CreateColorRGB(1, 0, 0), 1, 0, 0, Vertex(1)

Dx.CreateD3DLVertex 0, 0, 0, _
Dx.CreateColorRGB(0, 1, 0), 1, 0, 0, Vertex(2)

Dx.CreateD3DLVertex 10, 0, 0, _
Dx.CreateColorRGB(0, 0, 1), 1, 0, 0, Vertex(3)
```

ثم نبدأ في إنشاء ماتركس العالم MatWorld كالتالي :

```
Dx.IdentityMatrix MatWorld
```

ثم نحدد للسواقة Device أن هذا هو ماتركس العالم :

```
Device.SetTransform D3DTRANSFORMSTATE_WORLD, MatWorld
```

بعد هذا ننشئ ماتركس العنصر :

```
Dx.ProjectionMatrix MatProj, 1, 100, PI / 2#
```

فأما أو متغير بعد MatProj فهو يعبر عن اقرب مسافة يمكن تصوير العنصر منها , وأما الثاني فيعبر عن أبعد مسافة , أم الثالث فيعبر عن توازي الجسم مع الأرض .

ثم نخبر السواقة أن هذا هو ماتركس العنصر :

```
Device.SetTransform D3DTRANSFORMSTATE_PROJECTION,
MatProj
```

بعد هذا ننشئ ماتركس الكاميرا :

```
Dx.ViewMatrix MatView, MakeVector(10, 20, 0),
MakeVector(0, 0, 0), MakeVector(0, 1, 0), 0
```

فأما أول Vector فيعبر عن مكان الكاميرا , وأما ثاني Vector فيعبر عن المكان الذي تنظر إليه الكاميرا , وأما ثالث Vector فيعبر عن زاوية الرؤية , وأما رابع متغير , فيعبر عن زاوية دوران الكاميرا .

بعد هذا نخبر السواقة أن هذا هو ماتركس الكاميرا , كالتالي :

```
Device.SetTransform D3DTRANSFORMSTATE_VIEW, MatView
```

مادمنا نتعامل مع النقاط حتى الآن , فنحن لسنا بحاجة للإضاءة , لذا نخبر السواقة بأن لا تنشئ أي إضاءات :

```
Device.SetRenderState D3DRENDERSTATE_LIGHTING, 0
```

صفر هنا تعني False , كذلك نخبر السواقة بأننا نريد الـ Shade Mode بأن يكون D3DSHADE_GOURAUD حيث أن Shade Mode تتحكم في شكل ظهور العناصر , وأفضل استخداماتها هي D3DSHADE_GOURAUD :

```
Device.SetRenderState D3DRENDERSTATE_SHADEMODE,
D3DSHADE_GOURAUD
```

بعد هذا نعلن عن D3DRECT المستخدم في عمل CLS للسواقة كالتالي :

```
Dim ClearRect(0 To 0) As D3DRECT
```

ثم نعين خواصه , تبعا لإحداثيات الشاشة , كالتالي :

```
ClearRect(0).X2 = 800
ClearRect(0).Y2 = 600
```

ثم نعلن عن متغير سيستخدم في تحديد زاوية دوران ماتركس العالم , كالتالي :

```
Dim R As Integer
```

هكذا البرنامج جاهز للعرض على الشاشة , نبدأ الحلقة التكرارية :

```
Do
```

نقلل قيمة زاوية دوران العالم , حتى لو وصلت إلى الصفر , تبدأ من ٣٦٠ مرة أخرى , وإذا كنت تريد أن تجعل العالم يدور في الاتجاه المضاد , فعليك جعل الزاوية تزيد , لا تقل :

```
R = R - 1

If R <= 0 Then R = 360
```

بعد هذا نأمر ماتركس العالم أن يدور على المحور Y بمقدار الزاوية , والمعروف أنه بدوران العالم , فإنه كل الكائنات بداخله تدور , كالتالي :

```
Dx.RotateYMatrix MatWorld, R * (Rad)
```

ويمكن أن نجعله يدور على المحور X لو غيرنا حرف Y من RotateYMatrix إلى حرف X , والـ Z كذلك .

بعد هذا نخبر السواقة أن العالم قد دار حتى تقوم بتجديد احداثياتها :

```
Device.SetTransform D3DTRANSFORMSTATE_WORLD, MatWorld
```

بعد هذا نقوم بعمل CLS للسواقة , باستخدام الأمر Clear كالتالي :

```
Device.Clear 1, ClearRect, D3DCLEAR_TARGET, 0, 0, 0
```

نخبر السواقة أننا نستعد للرسم :

```
Device.BeginScene
```

نبدأ الرسم :

```
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_LVERTEX, Vertex(0), 4, D3DDP_WAIT
```

نخبر السواقة أننا أنهينا الرسم :

```
Device.EndScene
```

نأمر السطح الرئيسي بإظهار الرسومات :

```
Primary.Flip Nothing, DDFLIP_WAIT
```

ننه الحلقة التكرارية :

```
Loop
```

رسم المجسمات

كما قلت من قبل , فإن أبسط وحدة لتكوين السطوح هي المثلث , في حين أن أبسط وحدة (الوحدة البنائية) لتكوين المجسمات هب المربعات Rectangles , أو المستطيلات للدقة اللغوية , وعلى هذا يمكننا رسم مجسم , مكعب مثلا , عن طريق رسم ستة سطوح , تمثل أوجه المكعب .

تعال نبدأ مشروع جديد , ونضيف إليه الوحدة البرمجية Module التي استخدمناها في المثال السابق .

سنستخدم ستة أوجه لرسم المكعب , ولما كان كل وجه عبارة عن شكل رباعي , يتكون من أربعة رؤوس , فإن إجمالي النقط التي سنستخدمها هي ٢٤ نقطة .

الآن نذهب إلى صفحة الكود الخاص بالفورم , وفيه نعلن عن المتغيرات الرئيسية , كالتالي :

```
Dim vTop(3) As D3DLVERTEX
```

لرسم الوجه الأعلى

```
Dim vBottom(3) As D3DLVERTEX
```

الوجه الأسفل

```
Dim vLeft(3) As D3DLVERTEX
```

الوجه اليسار

```
Dim vBack(3) As D3DLVERTEX
```

الوجه الخلفي

```
Dim vRight(3) As D3DLVERTEX
```

الوجه اليمين

```
Dim vFront(3) As D3DLVERTEX
```

الوجه الأمامي , ونحن سنستخدم أربعة ألوان , لتلوين كل وجه هي الأبيض , والأحمر , والأخضر , والأزرق , وسنعلن عن أربعة متغيرات لتحمل قيم هذه الألوان كالتالي :

```
Dim cWhite As Long

Dim cRed As Long

Dim cGreen As Long

Dim cBlue As Long
```

الآن ننشئ مقطعا Sub جديدا , وسنستخدمه في تحديد أماكن النقاط الأربعة وعشرون , على المحاور X – Y – Z , كالتالي :

```
Sub Init_Vertex()


End Sub
```

ونضع فيه قيم الوجه الأمامي للمكعب كالتالي :

```
Dx.CreateD3DLVertex 2, -2, 0, cWhite, 1, 0, 0,

vFront(0)

Dx.CreateD3DLVertex -2, -2, 0, cRed, 1, 0, 0,

vFront(1)

Dx.CreateD3DLVertex 2, 2, 0, cGreen, 1, 0, 0,

vFront(2)
```

```
Dx.CreateD3DLVertex -2, 2, 0, cBlue, 1, 0, 0,
vFront(3)
```

ثم نضع قيمة الوجه الأعلى كالتالي :

```
Dx.CreateD3DLVertex -2, 2, 5, cRed, 1, 0, 0, vTop(0)
Dx.CreateD3DLVertex 2, 2, 5, cWhite, 1, 0, 0, vTop(1)
Dx.CreateD3DLVertex -2, 2, 0, cBlue, 1, 0, 0, vTop(2)
Dx.CreateD3DLVertex 2, 2, 0, cGreen, 1, 0, 0, vTop(3)
```

ثم نضع قيم الوجه اليساري كالتالي :

```
Dx.CreateD3DLVertex 2, 2, 0, cBlue, 1, 0, 0, vLeft(0)
Dx.CreateD3DLVertex 2, 2, 5, cWhite, 1, 0, 0,
vLeft(1)
Dx.CreateD3DLVertex 2, -2, 0, cGreen, 1, 0, 0,
vLeft(2)
Dx.CreateD3DLVertex 2, -2, 5, cRed, 1, 0, 0, vLeft(3)
```

ثم نضع قيم الوجه الخلفي , كالتالي :

```
Dx.CreateD3DLVertex 2, 2, 5, cWhite, 1, 0, 0,
vBack(0)
Dx.CreateD3DLVertex -2, 2, 5, cRed, 1, 0, 0, vBack(1)
Dx.CreateD3DLVertex 2, -2, 5, cBlue, 1, 0, 0,
vBack(2)
Dx.CreateD3DLVertex -2, -2, 5, cGreen, 1, 0, 0,
vBack(3)
```

ثم الوجه اليميني :

```
Dx.CreateD3DLVertex -2, 2, 5, cBlue, 1, 0, 0,
vRight(0)
Dx.CreateD3DLVertex -2, 2, 0, cGreen, 1, 0, 0,
vRight(1)
Dx.CreateD3DLVertex -2, -2, 5, cWhite, 1, 0, 0,
vRight(2)
Dx.CreateD3DLVertex -2, -2, 0, cRed, 1, 0, 0,
vRight(3)
```

ثم الوجه الأسفل :

```
Dx.CreateD3DLVertex -2, -2, 0, cRed, 1, 0, 0,
vBottom(0)
Dx.CreateD3DLVertex 2, -2, 0, cGreen, 1, 0, 0,
vBottom(1)
Dx.CreateD3DLVertex -2, -2, 5, cBlue, 1, 0, 0,
vBottom(2)
Dx.CreateD3DLVertex 2, -2, 5, cWhite, 1, 0, 0,
vBottom(3)
```

الآن نذهب إلى إجراء تحميل الفورم , ونبدأ بنداء الدالة المستخدمة في إنشاء كائنات Direct3D , والموجودة ضمن الوحدة البرمجية التي أضفناها , كالتالي :

```
Init_Direct3DIM 600, 800, 16
```

ثم نحدد قيم الألوان الأربعة التي أعلنا عنها :

```
cWhite = Dx.CreateColorRGB(1, 1, 1)
cRed = Dx.CreateColorRGB(1, 0, 0)
```

```
cGreen = Dx.CreateColorRGB(0, 1, 0)

cBlue = Dx.CreateColorRGB(0, 0, 1)
```

ثم نقوم بنداء الدالة المستخدمة في تعيين مواقع النقط الأربعة وعشرون , والتي أنشأناها توا , وكالتالي :

```
Init_Vertex
```

نبدأ في إنشاء World Matrix كالتالي :

```
Dx.IdentityMatrix MatWorld

Device.SetTransform D3DTRANSFORMSTATE_WORLD, MatWorld
```

ثم ننشئ Projection Matrix كالتالي :

```
Dx.ProjectionMatrix MatProj, 1, 100, PI / 2

Device.SetTransform D3DTRANSFORMSTATE_PROJECTION,

MatProj
```

ثم ننشئ View Matrix كالتالي :

```
Dx.ViewMatrix MatView, MakeVector(0, 10, -10),
MakeVector(0, 0, 0), MakeVector(0, 1, 0), 0
Device.SetTransform D3DTRANSFORMSTATE_VIEW, MatView
```

نحدد للسواقة Device طريقة عرض المجسمات التي نريدها :

```
Device.SetRenderState D3DRENDERSTATE_LIGHTING, 0
Device.SetRenderState D3DRENDERSTATE_SHADEMODE,
D3DSHADE_GOURAUD
```

نعلن عن , وننشئ D3DRECT الخاص بمسح السواقة (هو يقوم بمسح الرسومات من على السطح الخفي) :

```
Dim ClearRect(0 To 0) As D3DRECT
ClearRect(0).X2 = 800
ClearRect(0).Y2 = 600
```

نعلن عن متغير سيكون هو زاوية دوران World Matrix :

```
Dim R As Integer
```

نبدأ الآن الحلقة التكرارية :

```
Do
```

نقوم بتحديد قيمة الزاوية :

```
R = R - 1
If R <= 0 Then R = 360
```

نقوم بتدوير World Matrix على المحور Y , تبعا للزاوية :

```
Dx.RotateYMatrix MatWorld, R * (Rad)
```

نقوم بإخبار السواقة أن World Matrix قد تغير :

```
Device.SetTransform D3DTRANSFORMSTATE_WORLD, MatWorld
```

نقوم بمسح السواقة :

```
Device.Clear 1, ClearRect, D3DCLEAR_TARGET, 0, 0, 0
```

نخبر السواقة أننا سنبدأ الرسم :

```
Device.BeginScene
```

نرسم الوجه الأمامي للمكعب :

```
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_LVERTEX, vFront(0), 4, D3DDP_WAIT
```

نرسم الوجه الأعلى للمكعب :

```
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_LVERTEX, vTop(0), 4, D3DDP_WAIT
```

نرسم الوجه الأيسر للمكعب :

```
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_LVERTEX, vLeft(0), 4, D3DDP_WAIT
```

نرسم الوجه الخلفي للمكعب :

```
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_LVERTEX, vBack(0), 4, D3DDP_WAIT
```

نرسم الوجه الأيمن للمكعب :

```
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_LVERTEX, vRight(0), 4, D3DDP_WAIT
```

نرسم الوجه السفلي للمكعب :

```
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_LVERTEX, vBottom(0), 4, D3DDP_WAIT
```

نخبر السواقة أننا أنهينا الرسم :

```
Device.EndScene
```

نأمر السطح الرئيسي بإظهار كل ما تم رسمه على السطح الخلفي :

```
Primary.Flip Nothing, DDFLIP_WAIT
```

ننه الحلقة التكرارية :

```
Loop
```

هكذا قمنا برسم مكعب , وهو شكل بسيط , ولكنه يوضح لك أساس رسم أي شيء , وأي مجسم معقد , وكذلك أخذنا فكرة عن كيفية رسم الوجه الواحد , أيا كان اتجاهه , وقد خصصت المثال الخامس عشر , من مجلد الأمثلة , لتوضيح لعمل المثال السابق .

الإكساء

إكساء المجسمات يعتبر أحد المهام المعقدة , والتي يتعب فيها المبرمجين كثيرا , ولكن إذا فهمت نظرية الإكساء , ومع بعض الممارسة , ستجد الإكساء عملية سهلة , وبسيطة جدا , فما هو الإكساء ؟؟؟

قمنا بعمل مكعب , في الدروس السابقة , ولونا أطرافه , أو رؤوسه , وعن طريق هذه الرؤوس , تم تلوين أوجه المكعب , وهذا يعد تلوينا , أم الإكساء , فهو تغطية

أحد أو بعض أو كل هذه الرؤوس , بصورة خارجية , من النوع النقطي Bit Map بحيث يتغير حجم , وشكل الصورة , لتنطبق على الوجه الذي تم إكساؤه .

وقد استخدمنا في الأمثلة السابقة النقاط من نوع D3DTLVERTEX و D3DLVERTEX , أما مع الإكساء , فيفضل استخدام D3DVERTEX .

سنرسم ذات المكعب الذي قمنا برسمه في المثال السابق , ولكن باستخدام النقط D3DVERTEX , ثم نكسي أوجهه الستة , بصورة Texture .

أبدأ مشروع جديد , وأضف له الوحدة البرمجية Module التي استخدمناها في الأمثلة السابقة , ثم اذهب إلى صفحة كود الفورم , وفيها أعلن عن المتغيرات الرئيسية كالتالي :

```
Dim vTop(3) As D3DVERTEX
Dim vBottom(3) As D3DVERTEX
Dim vLeft(3) As D3DVERTEX
Dim vBack(3) As D3DVERTEX
Dim vRight(3) As D3DVERTEX
Dim vFront(3) As D3DVERTEX

Dim Texture As DirectDrawSurface7
```

```
Dim Ddsd2 As DDSURFACEDESC2


Dim I As Long
```

نلاحظ من السابق , أن الإكساء عبارة عن DirectDrawSurface7 عادي , من المستخدم في Direct Draw , وكذلك كائن تعيين خصائصه من نوع Ddsurfacedesc2 , وهو أيضا الذي استخدمناه في Direct Draw .

ثم نذهب إلى إجراء Form_Load , ونأمر البرنامج بإنشاء كائنات Direct3D كالتالي :

```
Init_Direct3DIM 600, 800, 16
```

بعد هذا نقوم بنداء الدالة الخاصة بتحديد مواقع نقاط بناء المكعب , ثم دالة إنشاء الإكساء كالتالي :

```
Init_Vertex

CreateTexture
```

بعد هذا نقوم بإنشاء دالة تحديد مواقع نقاط بناء المكعب كالتالي :

```
Sub Init_Vertex()
```

```
' Create the front face
Dx.CreateD3DVertex 2, -2, 0, 0, 0, -1, 0, 0,
vFront(0)
Dx.CreateD3DVertex -2, -2, 0, 0, 0, -1, 1, 0,
vFront(1)
Dx.CreateD3DVertex 2, 2, 0, 0, 0, -1, 0, 1, vFront(2)
Dx.CreateD3DVertex -2, 2, 0, 0, 0, 1, 1, 1, vFront(3)

'Create the top face
Dx.CreateD3DVertex -2, 2, 5, 0, 0, -1, 0, 0, vTop(0)
Dx.CreateD3DVertex 2, 2, 5, 0, 0, -1, 1, 0, vTop(1)
Dx.CreateD3DVertex -2, 2, 0, 0, 0, -1, 0, 1, vTop(2)
Dx.CreateD3DVertex 2, 2, 0, 0, 0, 1, 1, 1, vTop(3)

'Create the left face
Dx.CreateD3DVertex 2, 2, 0, 0, 0, -1, 0, 0, vLeft(0)
Dx.CreateD3DVertex 2, 2, 5, 0, 0, -1, 1, 0, vLeft(1)
Dx.CreateD3DVertex 2, -2, 0, 0, 0, -1, 0, 1, vLeft(2)
Dx.CreateD3DVertex 2, -2, 5, 0, 0, 1, 1, 1, vLeft(3)
```

```
'create the back face
Dx.CreateD3DVertex 2, 2, 5, 0, 0, -1, 0, 0, vBack(0)
Dx.CreateD3DVertex -2, 2, 5, 0, 0, -1, 1, 0, vBack(1)
Dx.CreateD3DVertex 2, -2, 5, 0, 0, -1, 0, 1, vBack(2)
Dx.CreateD3DVertex -2, -2, 5, 0, 0, 1, 1, 1, vBack(3)

'create the right face
Dx.CreateD3DVertex -2, 2, 5, 0, 0, -1, 0, 0,
vRight(0)
Dx.CreateD3DVertex -2, 2, 0, 0, 0, -1, 1, 0,
vRight(1)
Dx.CreateD3DVertex -2, -2, 5, 0, 0, -1, 0, 1,
vRight(2)
Dx.CreateD3DVertex -2, -2, 0, 0, 0, 1, 1, 1,
vRight(3)

'create the bottom face
Dx.CreateD3DVertex -2, -2, 0, 0, 0, -1, 0, 0,
vBottom(0)
```

```
Dx.CreateD3DVertex 2, -2, 0, 0, 0, -1, 1, 0,
vBottom(1)
Dx.CreateD3DVertex -2, -2, 5, 0, 0, -1, 0, 1,
vBottom(2)
Dx.CreateD3DVertex 2, -2, 5, 0, 0, 1, 1, 1,
vBottom(3)
End Sub
```

لاحظ هنا أن آخر ثلاثة متغيرات , من متغيرات دالة Dx.CreateD3Dvertex تتغير من نقطة إلى أخرى , على الوجه الواحد , وهذا يساعد البرنامج على بناء الإكساء على المكعب , ويمكنك زيادة , أو تقليل , نسبة الإكساء , بتغيير الرقم ١ إلى ٠,٥ , أو ٢ , مثلا .

الآن ننشئ الدالة الخاصة بإنشاء الإكساء , كالتالي :

```
Sub CreateTexture()
```

أو لا نعلن عن متغير سيحمل قيم الإكساء :

```
Dim tEnum As Direct3DenumPixelFormats
```

ثم نحدد قيم DDSURFACEDESC2 :

```
Ddsd2.lFlags = DDSD_CAPS Or DDSD_HEIGHT Or DDSD_WIDTH
Or DDSD_PIXELFORMAT Or DDSD_TEXTURESTAGE
```

ننشئ الكائن الذي أعلنا عنه :

```
Set tEnum = Device.GetTextureFormatsEnum
```

لابد أن تكون قيمة Ddsd2.ddpfPixelFormat.lRGBBitCount تساوي ١٦ , لذا فإننا نبحث عن كل القيم التي يوفرها Device بحثا عن القيمة التي تساوي ١٦ من قيم Ddsd2.ddpfPixelFormat.lRGBBitCount :

```
For I = 1 To tEnum.GetCount
tEnum.GetItem I, Ddsd2.ddpfPixelFormat

If Ddsd2.ddpfPixelFormat.lRGBBitCount = 16 Then Exit
For

Next
```

الآن نضع جملة شرطية للتأكد من أن

Ddsd2.ddpfPixelFormat.lRGBBitCount = 16 , فإن لم تكن تساوي

١٦ , يقوم البرنامج بإنهاء نفسه :

```
If Ddsd2.ddpfPixelFormat.lRGBBitCount <> 16 Then End
```

نكمل بناء Ddsd2 كالتالي :

```
Ddsd2.ddsCaps.lCaps = DDSCAPS_TEXTURE
Ddsd2.ddsCaps.lCaps2 = DDSCAPS2_TEXTUREMANAGE
Ddsd2.lTextureStage = 0
```

ننشئ الإكساء Texture بناء على Ddsd2 :

```
Set Texture = Dd.CreateSurfaceFromFile(App.Path &
"\Texture.bmp", Ddsd2)
```

ننه الدالة :

```
End Sub
```

هذه هي دالة إنشاء الكسوة , والآن نعود إلى إجراء تحميل الفورم , ونتابع فيه العمل , الآن سنقوم بإنشاء الماتريكسات Matrices , كما تعلمنا في الدروس السابقة :

```
Dx.IdentityMatrix MatWorld
Device.SetTransform D3DTRANSFORMSTATE_WORLD, MatWorld

Dx.ProjectionMatrix MatProj, 1, 100, PI / 2
Device.SetTransform D3DTRANSFORMSTATE_PROJECTION,
MatProj

Dx.ViewMatrix MatView, MakeVector(0, 10, -10),
MakeVector(0, 0, 0), MakeVector(0, 1, 0), 0
Device.SetTransform D3DTRANSFORMSTATE_VIEW, MatView
```

الآن سنحدد قيمة RenderState للسواقة , بحيث تكون بدون إضاءة , وبحيث تعتمد على النظام D3DSHADE_GOURAUD في بناء المجسمات :

```
Device.SetRenderState D3DRENDERSTATE_LIGHTING, 0
```

```
Device.SetRenderState D3DRENDERSTATE_SHADEMODE,
D3DSHADE_GOURAUD
```

بعد هذا نقوم بالإعلان عن , وتحديد صفات الـ D3DRECT المكلف بمسح السطح الخلفي :

```
Dim ClearRect(0 To 0) As D3DRECT
ClearRect(0).X2 = 800
ClearRect(0).Y2 = 600
```

ثم نعلن عن المتغير الخاص بتحديد زاوية دوران الـ World Matrix :

```
Dim R As Integer
```

نبدأ الحلقة التكرارية , ونغير زاوية الدوران , ثم ندير World Matrix تبعا لهذه الزاوية , ثم نخبر السواقة بأننا أدرنا World Matrix كالتالي :

```
Do
R = R - 1
If R <= 0 Then R = 360
```

```
Dx.RotateYMatrix MatWorld, R * (Rad)



Device.SetTransform D3DTRANSFORMSTATE_WORLD, MatWorld
```

بعد هذا نقوم بمسح السطح الخلفي , ونخبر السواقة بأن تستعد للرسم :

```
Device.Clear 1, ClearRect, D3DCLEAR_TARGET, 0, 0, 0

Device.BeginScene
```

بعد هذا نخبر السواقة بأن ترسم Texture على السطوح التالية , وهو ما يسمى بالإكساء :

```
Device.SetTexture 0, Texture
```

الآن أي وجه سيتم رسمه , سيتم وضع الإكساء هذا عليه , وحتى توقف وضع الإكساء عند وجه معين , فإما أن تضع إكساء آخر , وإما تغير شكل الأمر ليكون كالتالي :

```
Device.SetTexture 0, NoThing
```

الآن نرسم الوجوه :

```
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, vFront(0), 4, D3DDP_WAIT

Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, vTop(0), 4, D3DDP_WAIT

Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, vLeft(0), 4, D3DDP_WAIT

Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, vBack(0), 4, D3DDP_WAIT

Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, vRight(0), 4, D3DDP_WAIT

Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, vBottom(0), 4, D3DDP_WAIT
```

الآن نقوم بإخبار السواقة بأننا أنهينا الرسم , ونخبر السطح الرئيسي بأن يظهر الرسوم , ونقفل الحلقة التكرارية :

```
Device.EndScene

Primary.Flip Nothing, DDFLIP_WAIT

Loop
```

هكذا نكون قد عرفنا كيف نكسي الوجوه , وبالتالي المجسمات , وقد وضعت المثال السادس عشر , من مجلد الأمثلة , كمثال على عمل الإكساء .

الإضاءة

في كل الأمثلة التي عملناها سابقا , في Direct3D , كنا نقوم دائما بإغلاق خاصية الإضاءة , ولو أننا لم نغلقها لما ظهرت لنا أي مجسمات , لأنه لا أضواء , نحن نعرف – في العالم الطبيعي – أنه لا يمكن أن نرى أي شيء إلا في مكان به ضوء , ولهذا لا نرى في الظلام , وهو الحال مع Direct3D فلو كنت قد قمت بتفعيل خاصية الإضاءة , بدون إضافة أضواء , فسيكون كأنك تقف في الظلام , ولن ترى شيئا .

ونحن سنشرح في هذا الدرس كيف تضيف الأضواء إلى مثالك , وكيف تضيئها , وتطفئها , وكيف تتحكم في ألوانها .

ابدأ مشروعا جديدا , وضع في الوحدة البرمجية التي نستخدمها في Direct3D , ثم اذهب إلى الفورم , وأكتب كودا يشبه تماما الكود الذي كتبناه في المثال السابق , ولكن بدون إضافة أي Device.SetRenderState , ليكون شكل كود الفورم كالتالي :

```
Private Sub Form_Load()
Init_Direct3DIM 600, 800, 16
Init_Vertex
CreateTexture
Dx.IdentityMatrix MatWorld
Device.SetTransform D3DTRANSFORMSTATE_WORLD, MatWorld

Dx.ProjectionMatrix MatProj, 1, 100, PI / 2
Device.SetTransform D3DTRANSFORMSTATE_PROJECTION,
MatProj
```

```
Dx.ViewMatrix MatView, MakeVector(0, 10, -10),
MakeVector(0, 0, 0), MakeVector(0, 1, 0), 0
Device.SetTransform D3DTRANSFORMSTATE_VIEW, MatView

Dim ClearRect(0 To 0) As D3DRECT
ClearRect(0).X2 = 800
ClearRect(0).Y2 = 600
Dim R As Integer
Do

R = R - 1
If R <= 0 Then R = 360

Dx.RotateYMatrix MatWorld, R * (Rad)

Device.SetTransform D3DTRANSFORMSTATE_WORLD, MatWorld
DoEvents
Device.Clear 1, ClearRect, D3DCLEAR_TARGET, 0, 0, 0
Device.BeginScene
```

```
Device.SetTexture 0, Texture
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, vFront(0), 4, D3DDP_WAIT

Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, vTop(0), 4, D3DDP_WAIT

Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, vLeft(0), 4, D3DDP_WAIT

Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, vBack(0), 4, D3DDP_WAIT

Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, vRight(0), 4, D3DDP_WAIT

Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, vBottom(0), 4, D3DDP_WAIT

Device.EndScene
```

```
Primary.Flip Nothing, DDFLIP_WAIT

Loop

End Sub
```

الآن اذهب إلى الجزء الذي يلي تعريف الماتريكسات مباشرة , ثم قم بالإعلان عن مادة Material جديدة , وهذه المادة بدونها لن يظهر المجسم :

```
Dim Mat7 As D3DMATERIAL7
```

ثم قم بتعيين خصائص هذه المادة , كالتالي :

```
With Mat7
.Ambient.b = 1: .Ambient.g = 1: .Ambient.R = 1
.diffuse.b = 1: .diffuse.g = 1: .diffuse.R = 1
End With
```

بعد هذا استخدم Device في تعيين هذه المادة , كالتالي :

```
Device.SetMaterial Mat7
```

ثم تعين RenderState :

```
Device.SetRenderState D3DRENDERSTATE_AMBIENT,
Dx.CreateColorRGBA(0.2, 0.2, 0.2, 0)
```

ويمكن زيادة شدة تأثير الضوء , من خلال زيادة الأرقام ما بين ٠ إلى ١ , حيث ١ هي أقوى إضاءة في حين ٠ هي الإظلام التام , بعد هذا نعلن عن كائن الإضاءة :

```
Dim Light7 As D3DLIGHT7
```

ثم نعين خصائصه , كالتالي :

```
With Light7
  .dltType = D3DLIGHT_POINT ' نوع الضوء
  .position = MakeVector(0, 10, -10) ' موقع (مصدر) الضوء
  .attenuation1 = 0.05
```

```
  .diffuse.a = 1 ' تعيين لون الضوء
  .diffuse.R = 1
  .diffuse.b = 1
  .diffuse.g = 1
  .range = 100 ' مدى تأثير الضوء
End With
```

نخبر السواقة عن الضوء :

```
Device.SetLight 0, Light7
```

الأمر التالي يقوم بتشغيل الضوء :

```
Device.LightEnable 0, True
```

ولإطفاؤه تغير القيمة True إلى False , والآن نعيد وضع RenderState :

```
Device.SetRenderState D3DRENDERSTATE_LIGHTING, True
```

```
Device.SetRenderState D3DRENDERSTATE_SHADEMODE,
D3DSHADE_GOURAUD
```

الآن قم بتشغيل المشروع , ولاحظ النتيجة .. وستجد كل شيء على مايرام بإذن اللّه , عدا أمر واحد , وهو أن الضوء غير ثابت , والسبب أنه يدور مع MatWorld , ولذا يجب أن نتعلم إيقافه , ونجعله لا يدور مع MatWorld , ولكن أولا سنتعلم كيف نتعامل مع أكثر من مجسم في البرنامج ...

لو وضعنا أكثر من مجسم في البرنامج , ثم أدرنا World Matrix باستخدام Dx.RotateYMatrix , فستدور هذه العناصر جميعها كأنها مرتبطة في بعضها , أو كأنها جسم واحد , ونحن نريد أن ندير كل منهما على حدة , في إتجاه مختلف عن الآخر , وهذا هو موضوع الدرس القادم .

العناصر المتعددة

تعلمنا أن World Matrix يعني الماتركس الخاص بالعالم ثلاثي الأبعاد الذي أنشأته بالكامل , ولما كان كل ما وضعناه حتى الآن في العالم ثلاثي الأبعاد مجرد مجسم واحد , فإننا بتدوير World Matrix فإن كل المجسمات بداخله تدور .

ولما كانت اللعبة , أقل لعبة , تتكون من أكثر من مجسم , فإننا يجب أن ننشئ World Matrix لكل مجسم على حدة .

ومن هذا نستنتج أن الماتريكس لا يتحكم فقط في العالم , والكاميرا , ولكنه يتحكم في كل مجسم أيضا , ولو كنا نريد أن ننشئ مجسمين مرتبطين ببعضهما البعض , فإننا نفعل هذا باستخدام ماتريكس واحد , ولو كنا سنجعل كل مجسم منفصل , فإننا نجعل لكل مجسم ماتريكس خاص به , وهذا مفيد جدا في عمل المجسمات المعقدة , كالسيارة مثلا , فإننا نصمم كل جزء منها على حدة , ثم نضيف الأجزاء جميعا إلى ماتريكس واحد , حتى تبدو للمستخدم , كأنها مجسم واحد .

نبدأ مشروعا جديدا , وفيه نضيف الوحدة البرمجية التي نستخدمها في تعريف كائنات Direct3D , ثم نذهب إلى صفحة الكود الخاص بالفورم , وفيه نعلن عن المتغيرات الرئيسية , كالتالي :

```
Dim RECT1(3) As D3DVERTEX

Dim RECT2(3) As D3DVERTEX


Dim Texture As DirectDrawSurface7

Dim Ddsd2 As DDSURFACEDESC2


Dim I As Long
```

فأما Rect1 فسيستخدم في رسم السطح (الوجه) الأول , وأما Rect2 فسيستخدم في رسم الوجه الثاني .

ونذهب إلى الإجراء الخاص بتحميل الفورم , ونأمر البرنامج بنداء الدالة التي تنشئ كائنات Direct3D , كالتالي :

```
Init_Direct3DIM 600, 800, 16
```

ثم نقوم بنداء الدالتين التين تنشئا المجسمان , الإكساء , كالتالي :

```
Init_Vertex

CreateTexture
```

ودالة الإكساء ستكون كما هي بدون تغيير , أما دالة تعيين مكان النقاط , فستتغير لترسم مجسمين , أحدهما يكون سطح افقي , والآخر يكون سطح رأسي :

```
Sub Init_Vertex()
Dx.CreateD3DVertex -2, 0, 2, 0, 0, -1, 0, 0, RECT1(0)
Dx.CreateD3DVertex 2, 0, 2, 0, 0, -1, 1, 0, RECT1(1)
Dx.CreateD3DVertex -2, 0, -2, 0, 0, -1, 0, 1,
RECT1(2)
Dx.CreateD3DVertex 2, 0, -2, 0, 0, 1, 1, 1, RECT1(3)

Dx.CreateD3DVertex -2, 2, 0, 0, 0, -1, 0, 0, RECT2(0)
Dx.CreateD3DVertex 2, 2, 0, 0, 0, -1, 1, 0, RECT2(1)
Dx.CreateD3DVertex -2, -2, 0, 0, 0, -1, 0, 1,
RECT2(2)
Dx.CreateD3DVertex 2, -2, 0, 0, 0, 1, 1, 1, RECT2(3)
End Sub
```

بعد هذا ننشئ الماتريكسات , والأضواء , والمواد , والريكت الخاص بمسح الرسومات من على السطح الخلفي , كما في المثال السابق تماما , ثم قبل أن نبدأ الحلقة التكرارية , نعلن عن متغيرين , من نوع الماتريكس , الأول سيمثل السطح الأول , والثاني سيمثل السطح الثاني :

```
Dim Mat1 As D3DMATRIX

Dim Mat2 As D3DMATRIX
```

نبدأ الحلقة التكرارية , ونغير قيمة الزاوية التي سيدور بها الماتريكسات :

```
Do


R = R - 1

If R <= 0 Then R = 360
```

بعد هذا ندير الماتريكسان , الأول على المحور Y والثاني على المحور Z , وبقيمة الزاوية :

```
Dx.RotateYMatrix Mat1, Rad * R

Dx.RotateZMatrix Mat2, Rad * R
```

نقوم بمسح السطح الخلفي , نخبر السواقة أننا سنبدأ الرسم :

```
Device.Clear 1, ClearRect, D3DCLEAR_TARGET, 100, 0, 0
Device.BeginScene
```

نخبر السواقة أن تكسي المجسمات بالإكساء الذي أنشأناه :

```
Device.SetTexture 0, Texture
```

نخبر السواقة أننا سنستخدم الماتريكس الأول في رسم السطح الأول , ثم نرسم السطح الأول :

```
Device.SetTransform D3DTRANSFORMSTATE_WORLD, Mat1
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, RECT1(0), 4, D3DDP_WAIT
```

نقوم بنفس الخطوة السابقة مع السطح الثاني :

```
Device.SetTransform D3DTRANSFORMSTATE_WORLD, Mat2
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, RECT2(0), 4, D3DDP_WAIT
```

ثم نخبر السواقة أننا أنهينا الرسم , ونخبر السطح الرئيسي بإظهار الرسومات , ثم ننه الحلقة التكرارية :

```
Device.EndScene

Primary.Flip Nothing, DDFLIP_WAIT

Loop
```

الملاحظ من السابق أن :

١-الشكل المرسوم يتبع الماتريكس المرسوم من خلاله

٢-يمكن استخدام ماتريكس واحد لأكثر من مجسم

٣-يمكن استخدام مجسم واحد لأكثر من ماتريكس

وقد خصصت المثال الثامن عشر , من مجلد الأمثلة , لتوضيح كيفية وضع أكثر من ماتريكس , في البرنامج الواحد .

التحريك

قلنا أن الماتريكس يتحكم في ظهور المجسمات , من حيث المكان Position ومن حيث الدوران Rotation , ومن حيث التحجيم Scale , فأما تدوير الماتريكس , فيتم من خلال DirectX نفسها كالتالي :

الدوران على المحور X :

```
Dx.RotateXMatrix Matrix, Angel
```

الدوران على المحور Y :

```
Dx.RotateYMatrix Matrix, Angel
```

الدواران على المحور Z:

```
Dx.RotateZMatrix Matrix, Angel
```

هذا عن الدوران , فأما التحريك فيتم عن طريق الأعضاء Rc41 و Rc42 و Rc43 , من أعضاء الماتريكس , ويتم التحريك كالتالي :

التحريك على المحور X :

```
Matrix.rc41 = X
```

والتحريك على المحور Y :

```
Matrix.rc42 = Y
```

والتحريك على المحور Z :

```
Matrix.rc43 = Z
```

وهكذا يتم التحريك , أما عن التحجيم Scaling فيتم من خلال الأعضاء Rc11 و Rc22 و Rc33 , من أعضاء Matrix , ويتم التحجيم كالتالي :

التحجيم على المحور X :

```
Matrix.rc11 = X
```

والتحجيم على المحور Y :

```
Matrix.rc22 = Y
```

والتحجيم على المحور Z :

```
Matrix.rc33 = Z
```

Matrix Multiply

من الملاحظ أن الماتريكس يستجيب لأمر واحد فقط , فمثلا لو أردت أن تدير المجسم على المحورين X و Y معا فإننا نكتب :

```
Dx.RotateXMatrix Matrix, Angel
Dx.RotateYMatrix Matrix, Angel
Device.SetTransform D3DTRANSFORMSTATE_WORLD, Matrix
```

ولكن لو أننا فعلنا هذا , فستلاحظ أن الماتريكس قد دار على المحور Y فقط , أي على المحور الأخير فقط من الكود , ولكي نجعل الماتريكس يستجيب لأكثر من أمر , فإننا نستعمل ماتريكس احتياطي , ثم نفعل التغييرات على الماتريكس

الاحتياطي , ثم ننقل هذه التغييرات من الماتريكس الإحتياطي إلى الماتريكس الرئيسي :

أولا نحرك (ندير) الماتريكس الرئيسي :

```
Dx.RotateYMatrix MatWorld, R * (Rad)
```

ثم نحرك الماتريكس المؤقت :

```
Dx.RotateZMatrix TempMatrix, R * Rad
```

ثم نعمل Multiply بين الرئيسي , والمؤقت , بحيث يأخذ الرئيسي , صفات المؤقت :

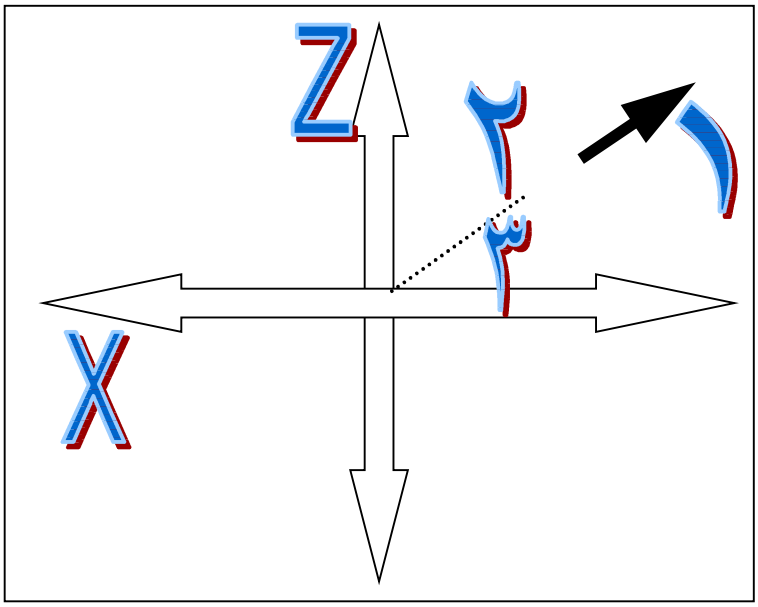
```
Dx.MatrixMultiply MatWorld, MatWorld, TempMatrix
```

ويمكننا أن ننفذ أكثر من تأثير , ثم نستخدم بعد كل تأثير Multiply لنقل هذا التأثير إلى الماتريكس الرئيسي .

الزوايا

تعرفنا حتى الآن على كيفية تحريك الماتريكسات Matrices , من خلال تغيير قيم أعضاء الماتريكس Rc41 و Rc42 و Rc43 , ولكن هذا لا يكفي , فنحن مثلا لو صنعنا سيارة , كمجسم , وضغطنا زر الأعلى , فإن السيارة تتحرك للأمام , ولو أدرنا السيارة مثلا جهة اليمين قليلا , ثم ضغطنا للأمام قليلا , فالمفروض أن تتحرك السيارة للأمام أيضا , ولكن الأمام بالنسبة للسيارة , وليس بالنسبة للكاميرا , أو World Matrix .

ويتم ذلك عن طريق معرفة زاوية دوران السيارة , وعن طريق هذه الزاوية , تعيين الإتجاه للأمام , وللخلف , بالنسبة للسيارة , وهذا الشكل يوضح لك ما أعنيه :



فأما السهم المشار إليه بالرقم (١) , فيمثل السيارة , والخط المتقطع خلفه , والمشار إليه بالرقم (٢) , وأما الزاوية التي يصنعها اتجاه حركة السيارة مع المحور الرئيسي للشاشة , فهي المشار إليها بالرقم (٣) , وعن طريق هذه الزاوية نستنتج اتجاه حركة السيارة , وبالتالي نستطيع جعلها تتحرك للأمام دائما , مهما كانت زاوية دورانها , ونحن نستنتج أولا زاوية الدوران , ثم نستنتج اتجاه السيارة , أولا نعلن عن ثلاثة متغيرات , الأول هو مقياس الزاوية , والثاني يحمل مقياس الزاوية على المحورين المتعامدين , (جيب الزاوية , وجيب تمام الزاوية) , وأما الثالث فسيحمل تأثير هذه الزاوية على اتجاه حركة المجسم , كالتالي :

```
Dim Angel As Integer
Dim Rotation As D3DVECTOR
Dim Position As D3DVECTOR
```

وإذا قمنا بتغيير الزاوية , فإننا نغير قيمة Vector تبعا للزاوية , مثلا :

```
Angel = Angel + 1
If Angel > 360 Then Angel = 1
Rotation.X = Cos(Angel * Rad)
Rotation.Z = Sin(Angel * Rad)
```

ونحن نعتمل على Sin و Cos الزاوية في التحريك , فالتحريك للأمام مثلا نغير قيم المتغير position كالتالي :

```
Position.X = Position.X - Rotation.X

Position.Z = Position.Z - Rotation.Z
```

وأما التحريك للخلف , فإننا نغير قيمة الإشارة السالبة , إلى موجبة .

تعال معا نتخذ مثالا على أساسيات التحريك التي تعرفنا عليها في الجزء السابق , نبدأ مشروعا جديدا , ونعلن عن المتغيرات الرئيسية , في كود الفورم , كالتالي :

```
Dim RECT1(3) As D3DVERTEX

Dim Land(3) As D3DVERTEX

Dim Texture As DirectDrawSurface7

Dim Texture2 As DirectDrawSurface7

Dim Ddsd2 As DDSURFACEDESC2

Dim Mat1 As D3DMATRIX


Dim Angel As Integer

Dim Rotation As D3DVECTOR
```

```
Dim Position As D3DVECTOR

Dim I As Long
```

ثم ننشئ المثال تماما كما في الأمثلة السابقة , ثم (قبل الحلقة التكرارية) نعلن عن الماتريكس المؤقت , المستخدم في عمل التأثيرات على World Matrix , كالتالي :

```
Dim TempMat As D3DMATRIX
```

ثم ننشئ دالة تحريك الماتريكس كالتالي :

```
Sub TranslateMatrix(Matrix As D3DMATRIX, X As Single,

Y As Single, Z As Single)

Dx.IdentityMatrix Matrix

With Matrix

.rc41 = X

.rc42 = Y

.rc43 = Z

End With
```

```
If X <> 0 Or Y <> 0 Or Z <> 0 Then Dx.ViewMatrix
MatView, MakeVector(0, 10, 0), Position,
MakeVector(0, 1, 0), 0
End Sub
```

السطر الأخير من الدالة يجعل الكاميرا تتابع السيارة (المتجه) دائما , أينما ذهبت , الآن نعود إلى الحلقة التكرارية :

نجعل الماتريكس يدور تبعا للزاوية :

```
Dx.RotateYMatrix Mat1, Angel * Rad
```

بعد هذا نجعل الماتريكس المؤقت يتحرك تبعا للموقع المفروض :

```
TranslateMatrix TempMat, Position.X, Position.Y,
Position.Z
```

بعد هذا نخبر السواقة عن أننا غيرنا اتجاه الكاميرا حتى تقوم بالتجديد

```
Device.SetTransform D3DTRANSFORMSTATE_VIEW, MatView
```

ثم ننقل تأثير الحركة من الماتريكس المؤقت إلى الماتريكس الرئيسي :

```
Dx.MatrixMultiply Mat1, Mat1, TempMat
```

الآن المثال جاهز للعمل , ولكن بقى عملية استخدام الكيبورد في تدوير , وتحريك السيارة , وذلك باستخدام Direct Input كما تعلمنا سابقا , كالتالي :

```
Sub GetKeys()
Didev.GetDeviceStateKeyboard DiKey
If DiKey.Key(DIK_DOWN) Then
Position.X = Position.X + Rotation.X
Position.Z = Position.Z + Rotation.Z
End If

If DiKey.Key(DIK_UP) Then
Position.X = Position.X - Rotation.X
Position.Z = Position.Z - Rotation.Z
End If

If DiKey.Key(DIK_LEFT) Then
```

```
Angel = Angel + 1

If Angel > 360 Then Angel = 1

Rotation.X = Cos(Angel * Rad)

Rotation.Z = Sin(Angel * Rad)

End If


If DiKey.Key(DIK_RIGHT) Then

Angel = Angel - 1

If Angel < 0 Then Angel = 359

Rotation.X = Cos(Angel * Rad)

Rotation.Z = Sin(Angel * Rad)

End If

End Sub
```

وقد خصصت المثل التاسع عشر , من مجلد الأمثلة , كمثال على التحريك الإحترافي .

الحدود , والتصادم , وتحريك الكاميرا

سنستخدم نسخة من المثال التاسع عشر , وهي المثال عشرين , من مجلد الأمثلة , لشرح عمليات (الحدود – التصادم – تحريك الكاميرا) .

أولا لجعل الكاميرا تتبع العنصر , أينما ذهب العنصر ذهبت وراؤه , وأعني هنا تحركت وراؤه , وليس مجرد تابعت اتجاهه , فمثلا قي ألعاب سباق السيارات , تكون الكاميرا الرئيسية خلف السيارة مباشرة , وتظل خلف السيارة كلما تحركت , أو دارت .

نستخدم زاوية التحريك , في معرفة الإتجاهات , عن طريق Sin و Cos هذه الزاوية , وعن طريق نفس الزاوية يمكننا وضع الكاميرا خلف العنصر تماما .

نستخدم Dx.ViewMatrix في تحديد موضع الكاميرا عن طريق المتغير الأول من متغيرات الدالة , والذي يسمى vFrom , وعلى هذا تكون شكل الدالة كالتالي :

```
Dx.ViewMatrix MatView,  MakeVector(Position.X +
Rotation.X * 5, 3, Position.Z + Rotation.Z * 5),
Position, MakeVector(0, 1, 0), 0
```

كذلك نستخدم المتغير Position في معرفة موضع المجسم , فلو كان أبعد من نقطة الحدود , فإننا نعيده إلى نقطة الحدود , كالتالي :

```
If Position.X < -18 Then Position.X = -18
If Position.Z < -19 Then Position.Z = -19
If Position.X > 18 Then Position.X = 18
If Position.Z > 19 Then Position.Z = 19
```

استخدام أكثر من View Port.

كلمة View Port تعني منفذ الرؤية , وهذا معناها اللفظي , الذي لا نحتاجه في شيء , ولكنها تعني , بالنسبة لنا , كمبرمجين , نافذة الرؤية , وكما وضحت سابقا , مثل مرآة السيارة في أي لعبة سباق سيارات , فهي عبارة عن كاميرا , تتجه عكس اتجاه السيارة , وتصور ما تراه خلف السيارة , ومنفذ رؤيتها , هو منطقة مرآة السيارة .

سنبدأ معا مثالا جديدا , ونضيف إليه الوحدة البرمجية التي نستخدمها مع Direct3D , ثم نذهب إلى صفحة كود هذه الوحدة , نجد في دالة إنشاء Direct3D الكود التالي :

```
Dim ViewPort As D3DVIEWPORT7
```

```
With ViewPort
  .lHeight = 600
  .lWidth = 400
  .lX = 0
  .lY = 0
  .maxz = 1
  .minz = 0
End With
Device.SetViewport ViewPort
```

نحذف هذه السطور السابقة , ثم نتجه إلى كود الفورم , وننشئ مثالا يشبه المثال السابق تماما , ثم نبدأ بوضع التغييرات كالتالي :

١-يتغير ClearRect ليكون كالتالي :

```
Dim ClearRect(0 To 0) As D3DRECT
ClearRect(0).X2 = 400
ClearRect(0).Y2 = 600
```

٢-نعلن عن ClearRect آخر , ليخصص لمنفذ الرؤية الآخر كالتالي :

```
Dim Clearrect2(0 To 0) As D3DRECT
Clearrect2(0).X2 = 800
Clearrect2(0).X1 = 400
Clearrect2(0).Y1 = 0
Clearrect2(0).Y2 = 600
```

ثم نعلن عن ماتريكس للكاميرا الأخرى :

```
Dim Camera2 As D3DMATRIX
```

نحن نعرف أن الكاميرا الأولى هي MatView , ثم نعلن عن View Port الأول :

```
Dim ViewPort As D3DVIEWPORT7
```

نحدد خصائصه بحيث يحتل نصف الشاشة الأيسر :

```
With ViewPort
 .lHeight = 600
 .lWidth = 400
```

```
  .lX = 0
  .lY = 0
  .maxz = 1
  .minz = 0
End With
```

ثم نعلن عن الثاني :

```
Dim ViewPort2 As D3DVIEWPORT7
```

ثم نعطيه نصف الشاشة الآخر :

```
With ViewPort2
  .lHeight = 600
  .lWidth = 400
  .lX = 400
  .lY = 0
  .maxz = 1
  .minz = 0
End With
```

الآن نتجه إلى الحلقة التكرارية , وقبل أن نكتب Device.Clear , نعين منفذ الرؤية الأول :

```
Device.SetViewport ViewPort
```

ثم نعين الكاميرا الأولى :

```
Device.SetTransform D3DTRANSFORMSTATE_VIEW, MatView
```

ثم نكتب كود بناء العناصر للمنفذ الأول :

```
Device.Clear 1, ClearRect, D3DCLEAR_TARGET, 0, 0, 0

Device.BeginScene
Device.SetTexture 0, Texture2
Device.SetTransform D3DTRANSFORMSTATE_WORLD, MatWorld
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_LVERTEX, Land(0), 4, D3DDP_WAIT
Device.SetTexture 0, Texture
```

```
Device.SetTransform D3DTRANSFORMSTATE_WORLD, Mat1
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, RECT1(0), 4, D3DDP_WAIT
Device.EndScene
```

بعد هذا نحدد مكان الكاميرا الأخرى , لتكون فوق العناصر (مشهد من أعلى) :

```
Dx.ViewMatrix Camera2, MakeVector(Position.X - 5, 10,
Position.Z + 4), Position, MakeVector(0, 1, 0), 0
```

نجعل السواقة تتعامل مع الكاميرا الثانية :

```
Device.SetTransform D3DTRANSFORMSTATE_VIEW, Camera2
```

ثم نعين منفذ الرؤية الثاني :

```
Device.SetViewport ViewPort2
```

بعد هذا نبدأ في رسم العناصر للمنفذ الثاني , تماما كما بالمنفذ الأول :

```
Device.Clear 1, Clearrect2, D3DCLEAR_TARGET, 0, 0, 0
```

```
Device.BeginScene
Device.SetTexture 0, Texture2
Device.SetTransform D3DTRANSFORMSTATE_WORLD, MatWorld
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_LVERTEX, Land(0), 4, D3DDP_WAIT
Device.SetTexture 0, Texture
Device.SetTransform D3DTRANSFORMSTATE_WORLD, Mat1
Device.DrawPrimitive D3DPT_TRIANGLESTRIP,
D3DFVF_VERTEX, RECT1(0), 4, D3DDP_WAIT
Device.EndScene
```

وقد خصصت المثال الحادي والعشرون من مجلد الأمثلة , كمثال للعمل مع أكثر من منفذ واحد للرؤية .

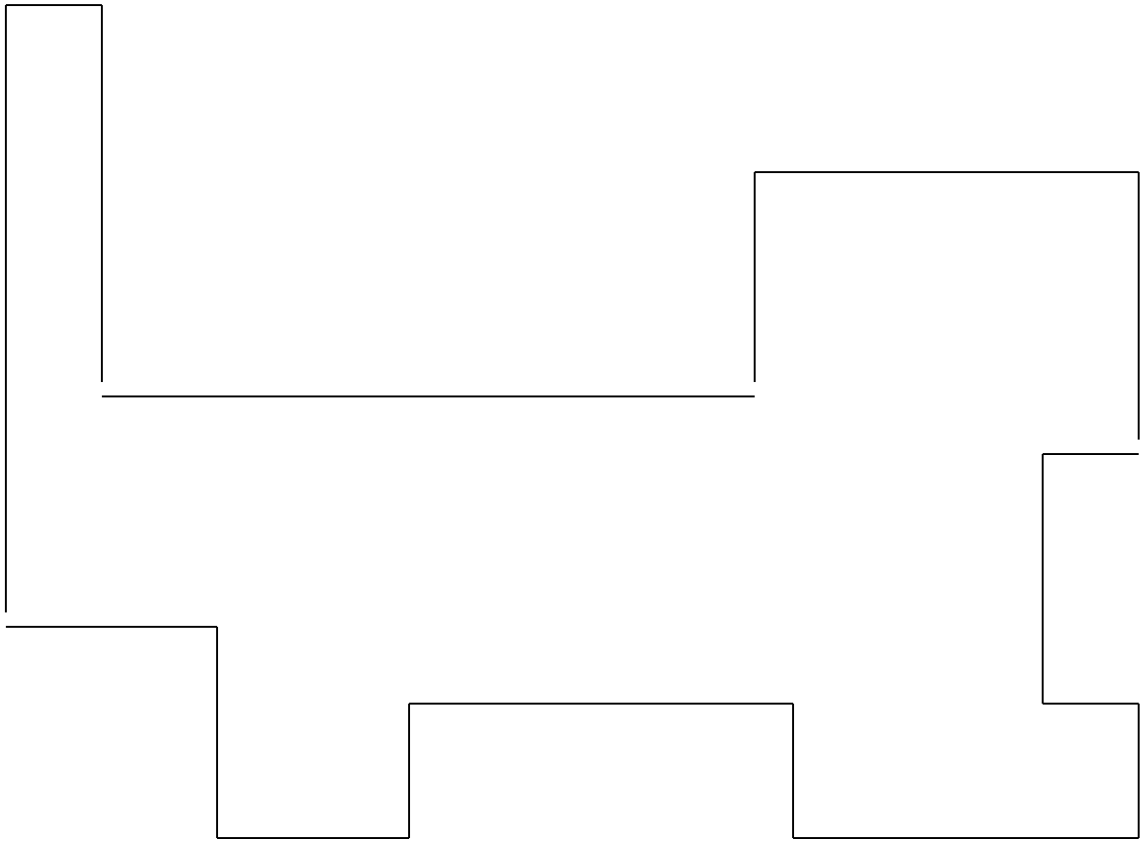
منطقة الخريطة

معنى مصطلح (منطقة الخريطة) أي تقسيم الخريطة إلى مناطق (أقسام) وذلك للعمل في الألعاب الكبيرة , والضخمة , والتي تشمل على خريطة كبيرة .

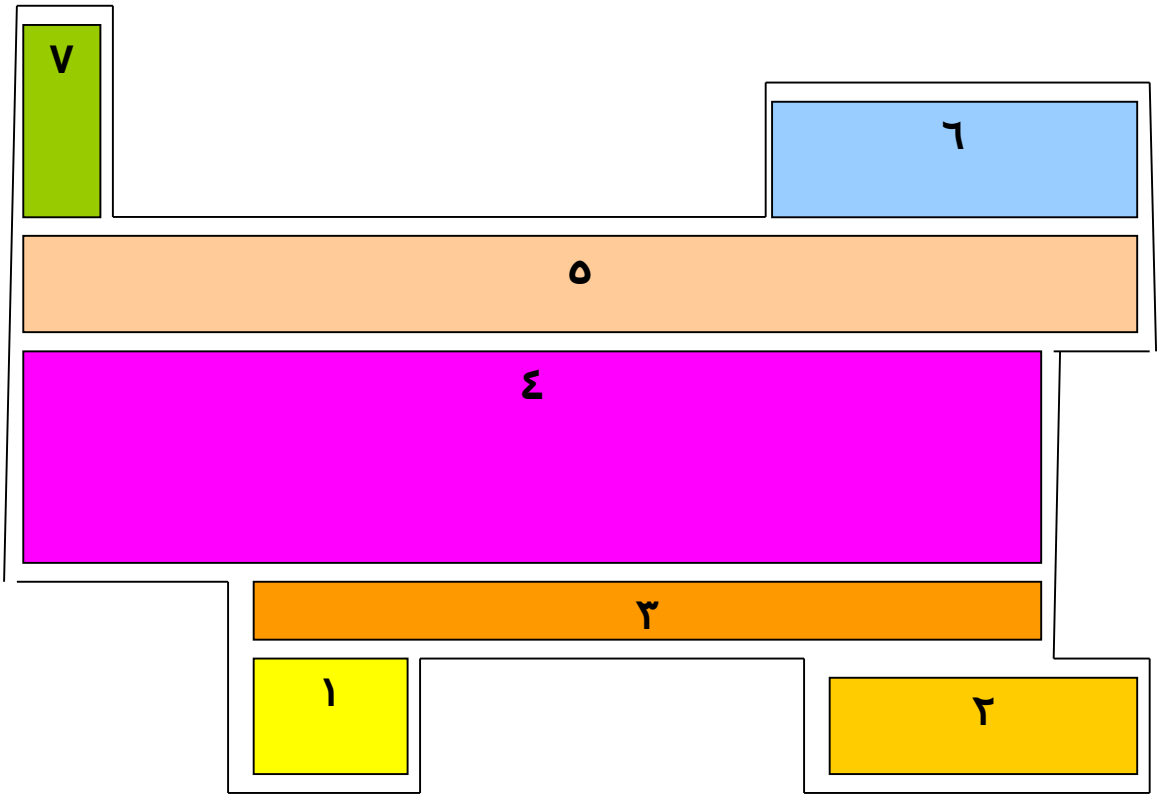
ومنطقة الخريطة تعني أنك تقسم الخريطة الكبيرة إلى مناطق أصغر , بحيث عندما يكون اللاعب موجودا في منطقة من مناطق الخريطة , يتعامل الكود مع

هذه المنطقة فقط , بدلا من أن يتعامل مع الخريطة الكاملة , وهذا سيسبب ابطا الكود كثيرا .

لنفترض أن عندك خريطة بهذا الشكل :



فإنه يمكن تقسيمها إلى مناطق مربعة , كالتالي :



وبالتالي , فإذا كان اللاعب في المنطقة رقم ٥ , فإن الكود يتعامل – من ناحية التصادم – مع المنطقة الخامسة فقط , فمن المستحيل أن يكون اللاعب مثلا في المنطقة الخامسة , ويصطدم بجسم , أو حائط في المنطقة رقم ١ , أو ٢ .

وطريقة تقسيم الخريطة إلى مربعات (أشكال رباعية) هي أسهل , وأفضل طريقة لتقسيم الخريطة , فمثلا لتخصص قسم للمنطقة رقم ١ , فأنت تحسب إحداثيات هذه المنطقة , ولتكن من النقطة (٠, ٠) إلى النقطة (٢٠, ٢٠) , فإذا كان Position اللاعب في هذه المنطقة , فليتعامل الكود مع المنطقة رقم واحد .

حساب سرعة اللعبة

سرعة اللعبة هي أهم مكون من مكونات اللعبة , وخاصة الألعاب ثلاثية الأبعاد , والتي قد تتأثر بالسرعة تأثرا كبيرا , نظرا لما تستخدمه المجسمات , والإكسائات , من مساحة في الذاكرة المؤقتة .

سرعة اللعبة لا تقاس بالمسافات , ولا بسرعة الدوران , لأن هذين النسبتين يمكن تغييرهما , بتغيير الأرقام , ولنفهم كيفية قياس السرعة في اللعبة تعال نتعرف على نظام عمل اللعبة .

اللعبة .. أي لعبة .. يبدأ كودها بإنشاء الكائنات الرئيسية , ثم تحديد مواقع المجسمات , وتعيين الإكساءات , وما إلى هذا من بناء اللعبة , ثم تبدأ الحلقة التكرارية , التي غالبا ما تكون Do أو Do While أو Do Until , وتسمى هذه الحلقة التكرارية Looping , وهي ساعة (Timer) اللعبة .

والحلقة التكرارية عندما تبدأ فهي تقوم بتنفيذ الأوامر المكتوبة داخل الحلقة التكرارية , حتى تصل إلى نهاية الحلقة التكرارية , ثم تعود مرة أخرى إلى بداية الحلقة التكرارية , وتظل تكرر هذه العملية حتى يتم الخروج من الحلقة التكرارية , أو الخروج من البرنامج .

وسرعة اللعبة تقاس بعدد تكرار أوامر الحلقة التكرارية في الثانية الواحدة , ومعظم الألعاب الموجودة حاليا تتميز بسرعة تصل من ١٦ إلى ٣٠ تكرار في الثانية .

وكل تكرار للحلقة التكرارية يسمى فرام Frame , وسرعة اللعبة تقاس بعدد الفرامات في الثانية الواحدة .

لو قلت سرعة لعبتك عن ٨ فرامات في الثانية فأنت بحاجة لتخفيف بعض الكائنات , أو حدة الإكساء في اللعبة , أما لو زادت عن ٦٠ فأعد النظر في لعبتك , فبالتأكيد هي أقل مما ينبغي من حيث القدرات الرسومية , أو قيمة اللعبة .

لحساب سرعة اللعبة أنت بحاجة إلى الإعلان عن ثلاثة متغيرات , وهي :

```
Dim LastCheck As Single
Dim Frames As Long
Dim fChange As Long
```

فأما LastChange فهو يعبر عن آخر وقت تم تغيير عدد الفرامات في الثانية الواحدة فيه , وهو يتغير كل ثانية , وأما frames فهو يتغير كلما انتهت الحلقة التكرارية وبدأت من جديد , وأما fChange فهو عدد الفرامات في الثانية الواحدة , وهو الرقم الذي يظهر للمستخدم .

أولا , نقوم بزيادة Frames , كلما بدأت الحلقة التكرارية من جديد , بهذا الأمر :

```
Frames = Frames + 1
```

إذا مرت ثانية كاملة , فإننا نحسب عدد الفرامات التي تغيرت في الثانية الكاملة , ثم نعيد عدد الفرامات إلى صفر لتبدأ من جديد , ثم نجعل LastChange تساوي التوقيت الحالي , حتى تبدأ الثانية من جديد :

```
If Dx.TickCount > LastCheck + 1000 Then
fChange = Frames
LastCheck = Dx.TickCount
Frames = 1
End If
```

بعد هذا نرسم FPS على الشاشة , باستخدام Back_Buffer :

```
Back_Buffer.DrawText 20, 20, CStr(fChange), False
```

وكمثال على تقسيم الخريطة , وكذلك على حساب سرعة اللعبة , فقد وضعت التطبيق السادس , من مجلد التطبيقات , لتوضيح هاتين العمليتين .

Direct3D RM Mode

نحن نعتبر الآن انتهينا من فصول Direct3D IM Mode , ولكننا سنعود له مرة أخرى , بعد فصول RM Mode , حتى نتعلم معا كيف نستخدم كلتا النوعين معا , في برنامج واحد .

يختلف Direct3D RM Mode عن Direct3D IM Mode في أنه أسرع , وأسهل في العمل , وكذلك أسهل في التعلم , كما يوفر لك ما يسمى بالفرامات , وهي تقليد لكائن Matrix , ولكنها تزداد عنه في عدد من الأشياء , وأخيرا يوفر لك RM Mode القدرة على التعامل مع المجسمات التي تم إنشاءها بواسطة البرامج ثلاثية الأبعاد مثل 3D Studio Max , والتي تم حفظها على النوع *.x , وهو نوع يمكن إنشاؤه بطريقة سهلة كما سنتعرف من خلال دروس الكتاب .

والتعامل مع ملفات *.X هذه هي لب التعامل مع Direct3D Rm Mode , وهو الشيء الوحيد الذي قد يدفعك إلى استخدام Direct3D Rm Mode .

يشبه أيضا Direct3D RM محركات الألعاب , حيث أنه لا يتيح لك التعامل مع العناصر مباشرة , ولكن يتيح لك التعامل مع العناصر عن طريق كائنات Direct3D RM .

كما توفر لك Direct3D RM Mode إمكانية عمل Texture Coordinates , وهي القدرة على تخصيص الإكساء , على السطح الواحد , مثل أن تجعل الإكساء يتكرر أكثر من مرة على السطح , أو يتكرر فقط على أحد محورين السطح , وهو أمر رائع كما سنرى في فصول الكتاب .

يوفر لك Direct 3D IM Mode القدرة على الدخول مباشرة إلى أوامر DirectX , ولهذا يفضلها المبرمجون المتمكنون , عن RM Mode , التي – كما قلت – تشبه التعامل من خلال محركات ألعاب , ويقوم هؤلاء المبرمجون باستخدام مكتبات خاصة خارجية , لتحميل المجسمات المعقدة , في Direct3D IM Mode , لأن هذا النوع لا يدعم قراءة الملفات التي تم صناعتها بواسطة برامج ثلاثية الأبعاد , ونحن يجب أن نتعلم التعامل مع النوعين , ولكن بدون استخدام مكتبات خارجية , فـ DirectX توفر لنا كل ما نريد , وكل ما سنفعله هو أن ندمج Direct3D Rm و Direct3D Im معا , بحيث يتيح لنا RM تحميل المجسمات , ويتيح لنا IM التعامل بشكل مباشر , وقوي , مع DirectX , وسنتعلم معا هذا من خلال فصول الكتاب .

Direct3D Retain Mode

الكائنات الرئيسية

كلمة Direct3D Rm Mode هي اختصار لـ Direct3D Retain Mode , أي – بترجمة حرفية – ديركت إكس المحفوظة , وأما Direct3D Im Mode فهي اختصار لـ Direct3D Immediate Mode فتعني دايركت إكس المباشرة .

سنتعرف معا على الكائنات الرئيسية , في Direct3D RM Mode :

Direct3DRM3

كائن Direct3D الرئيسي , وسنستخدمه بكثرة في هذا النوع , على عكس نظيره في النوع الآخر , الذي لم نستخدمه إلا مرة أو مرتين .

Direct3DRMDevice3

هذا الكائن يتحكم في سواقة عرض الفيديو , ومنه يمكننا إظهار / إخفاء الكائنات من على الشاشة , وكذلك التحكم في العالم ثلاثي الأبعد , والإضاءة , و ...

Direct3DRMViewport2

هذا هو كائن منفذ الرؤية , وهو شبيه بالكائن النظير له في Direct3D IM Mode , إلا أنه أكثر مرونة .

Direct3DRMFrame3

هذا هو لب تكوين العالم ثلاثي الأبعاد , وهو الكائن الذي يتحكم في كل شيء , من تحريك , وتدوير , وتحجيم , وإضاءة , وكل شيء , وهو شبيه بالكائن D3DMATRIX المستخدم في Direct3D IM , إلا أنه أكثر قوة , ويسمح بتضمين فرام داخل فرام .

Direct3DRMLight

عنصر الضوء , وهو عنصر هام جدا في البرنامج , أو اللعبة , ويعطيك عدد من الإمكانيات الجيدة , للتعامل مع الضوء .

Direct3DRMMeshBuilder3

يستخدم هذا الكائن في تحميل المجسمات التي صنعت بواسطة برامج ثلاثية الأبعاد , استعدادا لعرضها على الشاشة , ويتيح لك عمل تأثيرات على هذه الملفات .

Direct3DRMTexture3

هذا هو كائن الإكساء , فالإكساء في Direct3D RM لا يعتمد على كائن Direct Draw Surface 7 , وإنما يعتمد على هذا الكائن .

Direct3DRMFace2

قلنا من قبل أن كائن Direct3DRMMeshBuilder3 يستخدم في تحميل المجسمات من ملفات خارجية , والحقيقة أن هذه ليست وظيفته الوحيدة , فإن أي مجسم يظهر في اللعبة , سواء صنعته من ملف خارجي , أو كونته بنفسك , يجب أن يستخدم هذا الكائن للظهور على الشاشة , وكل مجسم يتكون من عدد من الأوجه , يمكن تمثيل كل وجه بأنه Face , أي كائن Direct3DRMFace2 .

D3DRMVERTEX

كما أن كل مجسم يتكون من عدد من الأوجه , فإن كل وجه يتكون من عدد من النقط , وكل نقطة من هذه النقط يمكن التعبير عنها بكائن D3DRMVERTEX .

Direct3DRMAnimationSet2

هذا الكائن موكل بتحميل الملفات التي صنعت بواسطة , برامج ثلاثية الأبعاد , وتحتوي على معلومات التحريك Animation Data حتى تتيح لك تحريك هذه المجسمات تماما كما كانت تتحرك في البرنامج ثلاثي الأبعاد .

هذه هي الكائنات التي نستخدمها من خلال Direct3D RM Mode , وأعتقد أنك قد مللت من هذه المقدمات , والتعريفات الطويلة , وأنك تنتظر أن نبدأ العمل فورا , وهذا ما سنفعله الآن .

يتعامل Direct3DrmMode مع DirectDraw4 وليس DirectDraw7 :

```
Dim Dx As New DirectX7
Dim Dd As DirectDraw4
Dim Primary As DirectDrawSurface4
Dim Back_Buffer As DirectDrawSurface4
Dim Ddsd As DDSURFACEDESC2
```

```
Dim D3d As Direct3DRM3

Dim Device As Direct3DRMDevice3
```

الآن نذهب إلى اجراء Form_Load , وفيه ننشئ Direct Draw :

```
Set Dd = Dx.DirectDraw4Create("")

Dd.SetCooperativeLevel Me.hWnd, DDSCL_ALLOWREBOOT Or

DDSCL_EXCLUSIVE Or DDSCL_FULLSCREEN

Dd.SetDisplayMode 800, 600, 16, 0, DDSDM_DEFAULT
```

بعد هذا ننشئ السطح الرئيسي :

```
Ddsd.lFlags = DDSD_CAPS Or DDSD_BACKBUFFERCOUNT

Ddsd.ddsCaps.lCaps = DDSCAPS_3DDEVICE Or

DDSCAPS_COMPLEX Or DDSCAPS_FLIP Or

DDSCAPS_PRIMARYSURFACE

Ddsd.lBackBufferCount = 1


Set Primary = Dd.CreateSurface(Ddsd)
```

بعد هذا ننشئ السطح الخلفي :

```
Ddsd.ddsCaps.lCaps = DDSCAPS_BACKBUFFER
Set Back_Buffer =
Primary.GetAttachedSurface(Ddsd.ddsCaps)
```

بعد هذا ننشئ كائن Direct3D :

```
Set D3d = Dx.Direct3DRMCreate
```

بعد هذا ننشئ كائن Device :

```
Set Device =
D3d.CreateDeviceFromSurface("IID_IDirect3DHALDevice",
Dd, Back_Buffer, D3DRMDEVICE_DEFAULT)
```

بعد ذلك نحدد صفات السواقة , حتى نعدها للعمل في برنامجنا :

```
Device.SetBufferCount 2
Device.SetQuality D3DRMLIGHT_ON Or D3DRMSHADE_GOURAUD
```

```
Device.SetTextureQuality D3DRMTEXTURE_NEAREST


Device.SetRenderMode

D3DRMRENDERMODE_BLENDEDTRANSPARENCY
```

الآن تم إعداد الكائنات الرئيسية في Direct3DRMMode , ولا يجب أن تنسى طريقة إنهاء البرنامج , أن تحذف كل هذه الكائنات من الذاكرة , كالتالي :

```
On Error Resume Next

Set Device = Nothing

Set Dd = Nothing

Set Dx = Nothing

Set D3d = Nothing

Set Primary = Nothing

Set Back_Buffer = Nothing

End
```

وقد خصصت المثال الثاني والعشرون , من مجلد الأمثلة , لشرح كيفية إنتاج الكائنات الرئيسية في Direct3D , وكذلك طريقة حذف هذه الكائنات من الذاكرة .

تكوين المجسمات

قبل أن نبدأ العمل الفعلي في Direct3D , سأشرح لك كيفية تكوين المجسمات التي يمكنك أن تعرضها في لعبتك .

أولا : تذهب إلى برنامج ثلاثي الأبعاد مثل 3D Studio Max , ثم تكون من خلال البرنامج المجسم الذي تريده (طائرة – سيارة – مقاتل ...) , وهذا يتطلب منك خبرة في إنشاء , وعمل المجسمات , ولو لم يكن لك خبرة , أو لا تملك برنامج ثلاثي الأبعاد , فاستعن بأصدقائك .

ثانيا : بعد تكوين المجسم , تصنع تصدير Export للمجسم , وتختر النوع *.3ds .

ثالثا : بعد هذا تذهب إلى برنامج Conv3ds.exe من خلال نافذة Dos , ولا تنس وضع المجسم , وبرنامج Conv3ds.exe في مجلد واحد .

رابعا : تكتب في الدوس الأمر التالي :

```
Conv3ds -m [اسم الملف ثلاثي الأبعاد]
```

و المعامل -m يعني أن المجسم من نوع Mesh , وهو النوع الذي نستخدمه في Direct3D .

وهناك بعض المعاملات الأخرى غير -m مثل :

-A

وضع معلومات التحريك Animation لو كنت أضمنت تحريك في الملف .

-M

عدم وضع معلومات المادة Material في الملف الناتج .

-S xxx

تعبر S عن التحجيم scaling وتعبر Xxx عن الأرقام , مثل ١٠ – ١٠٠ – ١١٢ وهكذا .

مثلا أنت صنعت المجسم , ثم حفظته باسم File1.3Ds , انسخه إلى نفس المجلد الذي يوجد فيه برنامج Conv3ds , ثم اذهب إلى نافذة DOS , وانتقل إلى المجلد الذي فيه البرنامج , والمجسم , ثم اكتب الأمر التالي :

```
conv3ds –m File1.3ds
```

سيقوم البرنامج بالتحميل قليلا , ثم بعد ذلك لو تفحصت المجلد , الذي فيه البرنامج , ستجد ملفا باسم File1.x وهذا هو الملف الذي يمكننا استخدامه في Direct3D .

برنامج Conv3Ds.exe مرفق مع الكتاب , في مجلد (خدميات) , في مجلد فرعي باسم (Conv3ds-dos) .

3D Converter

توفيرا للوقت , والجهد , فقد قمت بصنع برنامج للتحويل , يعمل من خلال بيئة النوافذ , وهو موجود في مجلد (خدميات) , في مجلد باسم (3D Converter) , وطريقة تشغيله كالتالي :

١-ضع البرنامج Conv3ds و برنامجي الذي صنعته , والملف الذي تريد تحويله في مجلد واحد .

٢-قم بتشغيل البرنامج 3D-Converter.exe

٣- في صندوق النص أكتب اسم الملف الذي تريد تحويله كالتالي :

File.3ds .

٤- اضغط على زر Convert ثم انتظر حتى يتم صنع الملف .

٥- ستجد ملفا بنفس اسم الملف الأصلي , ولكن بلاحقة *.x , قد تم انشاؤه في ذات المجلد .

والآن , وبعد أن تعلمنا كيف نعد المجسمات التي سنستخدمها في اللعبة , تعال نعمل على مثال لعرض مجسم من خلال لعبتك .

اذهب إلى برنامج ثلاثي الأبعد , وقم بتكوين مجسم , ثم احفظه على هيئة *.3ds , ثم قم بتحويله إلى *.X كما تعلمنا , وسمه Mesh.x , ولو لم يكن عندك برنامج ثلاثي الأبعاد , لتعد مجسما , يمكنك أن تأخذ المجسم المستخدم في المثال الثالث والعشرون , من مجلد الأمثلة , للعمل عليه .

ابدأ مشروعا جديدا , وفي قسم الإعلان العام في صفحة كود الفورم , أعلن عن المتغيرات الرئيسية كالتالي :

```
Dim Dx As New DirectX7

Dim Dd As DirectDraw4

Dim Primary As DirectDrawSurface4
```

```
Dim Back_Buffer As DirectDrawSurface4

Dim Ddsd As DDSURFACEDESC2


Dim D3d As Direct3DRM3

Dim Device As Direct3DRMDevice3

Dim Light As Direct3DRMLight

Dim Shadow As Direct3DRMLight

Dim WorldFrame As Direct3DRMFrame3

Dim CameraFrame As Direct3DRMFrame3

Dim LightFrame As Direct3DRMFrame3

Dim MeshFrame As Direct3DRMFrame3

Dim Mesh As Direct3DRMMeshBuilder3

Dim ViewPort As Direct3DRMViewport2
```

فأما Light و Shadow فهما كائنان لإنشاء الإضاءة , والظل , وأما كائن WorldFrame فهو كائن فرام العالم , وأما CameraFrame فهو الفرام المتحكم في الكاميرا , وأما LightFrame فهو الفرام المتحكم في الإضاءة , وأما MeshFrame فهو الفرام الذي سيعرض لنا المجسم , وأما Mesh فهو الكائن الذي سيحمل المجسم عليه , وأما ViewPort فقد تعرفنا عليه فيما سبق .

نذهب إلى إجراء تحميل الفورم وفيه نعلن عن متغيرات DirectDraw الرئيسية :

```
Set Dd = Dx.DirectDraw4Create("")
Dd.SetCooperativeLevel Me.hWnd, DDSCL_ALLOWREBOOT Or
DDSCL_EXCLUSIVE Or DDSCL_FULLSCREEN
Dd.SetDisplayMode 800, 600, 16, 0, DDSDM_DEFAULT
Show
Ddsd.lFlags = DDSD_CAPS Or DDSD_BACKBUFFERCOUNT
Ddsd.ddsCaps.lCaps = DDSCAPS_3DDEVICE Or
DDSCAPS_COMPLEX Or DDSCAPS_FLIP Or
DDSCAPS_PRIMARYSURFACE
Ddsd.lBackBufferCount = 1

Set Primary = Dd.CreateSurface(Ddsd)
Ddsd.ddsCaps.lCaps = DDSCAPS_BACKBUFFER
Set Back_Buffer =
Primary.GetAttachedSurface(Ddsd.ddsCaps)
```

ثم نعرف كائنات Direct3D كالتالي :

```
Set D3d = Dx.Direct3DRMCreate

Set Device =
D3d.CreateDeviceFromSurface("IID_IDirect3DHALDevice",
Dd, Back_Buffer, D3DRMDEVICE_DEFAULT)

Device.SetBufferCount 2
Device.SetQuality D3DRMLIGHT_ON Or
D3DRMRENDER_GOURAUD
Device.SetTextureQuality D3DRMTEXTURE_NEAREST

Device.SetRenderMode
D3DRMRENDERMODE_BLENDEDTRANSPARENCY
```

بعد هذا ننشئ فرام العالم :

```
Set WorldFrame = D3d.CreateFrame(Nothing)
```

ولأن كل الفرامات الأخرى تابعة إلى فرام العالم , فإننا ننشئ الفرامات الأخرى باستخدام فرام العالم :

```
Set CameraFrame = D3d.CreateFrame(WorldFrame)
Set LightFrame = D3d.CreateFrame(WorldFrame)
Set MeshFrame = D3d.CreateFrame(WorldFrame)
```

بعد هذا نحدد اللون العام للعالم , وغالبا ما يكون هذا اللون أسود , وحتى لو كان أسود , أو غير أسود , فأنت يجب أن تقوم بهذه الخطوة , وإلا فلن تظهر المجسمات :

```
WorldFrame.SetSceneBackgroundRGB 0.5, 0.5, 1
```

بعد هذا ننشئ ViewPort باستخدام فرام الكاميرا :

```
Set ViewPort = D3d.CreateViewport(Device,
CameraFrame, 0, 0, 800, 600)
```

والأرقام ٠ , ٠ , ٨٠٠ , ٦٠٠ تعبر عن إحداثيات الشاشة , وهي المساحة المستخدمة من قبل ViewPort , بعد هذا ننشئ الإضاءة , والظل :

```
Set Light = D3d.CreateLightRGB(D3DRMLIGHT_AMBIENT, 2,
2, 2)
Set Shadow = D3d.CreateLightRGB(D3DRMLIGHT_POINT, 1,
1, 1)
```

ثم نضيف الضوء , والظل , إلى الفرام الخاص بالضوء :

```
LightFrame.AddLight Light
LightFrame.AddLight Shadow
```

بعد هذا ننشئ الكائن الذي سيحمل المجسم :

```
Set Mesh = D3d.CreateMeshBuilder
```

ثم نحمل المجسم في الكائن :

```
Mesh.LoadFromFile App.Path & "\mesh.x", 0, 0,
Nothing, Nothing
```

ثم نغير حجم المجسم , وفي برنامجك يمكنك ألا تغير حجم المجسم , أو أن تغيره , حسب حجم المجسم عندك :

```
Mesh.ScaleMesh 0.1, 0.1, 0.1
```

بعد هذا نضيف الميش Mesh إلى الفرام المخصص له :

```
MeshFrame.AddVisual Mesh
```

نقوم بتغيير مكان المجسم , عن طريق الفرام :

```
MeshFrame.SetPosition Nothing, 0, 0, 20
```

بعد هذا نجعل الفرام يدور حول المحورين X و Y وبزاوية مقدارها ١ درجة :

```
MeshFrame.SetRotation Nothing, 1, 1, 0, 1
```

وبكتابة السطر السابق يظل المجسم يدور ولا يتوقف حتى تعطي له أمر الإيقاف كالتالي :

```
MeshFrame.SetRotation Nothing, 0, 0, 0, 0
```

ثم نبدأ الحلقة التكرارية :

```
Do
```

هذا الأمر يخبر فرام العالم أن يقوم بتجديد مواقع , ودورانات الفرامات التابعة له :

```
WorldFrame.Move 0.05
```

بعد هذا تخبر ViewPort أن يقوم تنظيف CLS السطح الخلفي :

```
ViewPort.Clear D3DRMCLEAR_TARGET Or
D3DRMCLEAR_ZBUFFER
```

ثم نأمر السواقة بتجديد السطح الخلفي :

```
Device.Update
```

بعد هذا نخبر ViewPort أن ترسم الكائنات التابعة لفرام العالم , على السطح الخلفي :

```
ViewPort.Render WorldFrame
```

ثم نخبر السطح لرئيسي برسم الرسومات التي على السطح الخلفي على الشاشة :

```
Primary.Flip Nothing, DDFLIP_WAIT
```

ثم نغلق الحلقة التكرارية :

```
DoEvents
Loop
```

وستجد المثال الثالث والعشرون من مجلد الأمثلة , يمثل لك كيفية إضافة مجسم إلى البرنامج .

قراءة المجسم

يجب أن تعرف أن المجسم مكون من وجوه Faces , وأن كل وجه مكون من وجهين عبارة عن مثلثين متلاقيين بالوتر , وكل مثلث مكون من ثلاثة رؤوس .

وقراءة المجسم تعني قراءة عدد وجوه المجسم , وعدد رؤوس المثلثات في المجسم , ومعرفة مكان كل رأس .

وهذا مفيد جدا , في عملك على تلوين أجزاء من المجسم بألوان خاصة , أو إكساء أجزاء من المجسم بإكساءات خاصة بكل جزء , وغير ذلك .. ستمكننا من التعامل مع المجسمات في Direct3D IM Mode كما سنرى معا .

يتم أولا الإعلان عن مصفوفة من الوجوه , ومصفوفة من الرؤوس , ومتغير سيحمل عدد من الوجوه , ومتغير سيحمل عدد من الرؤوس :

```
Dim fCount As Long, vCount As Long
Dim L As Long, L2 As Long
Dim Face() As Direct3DRMFace2
Dim Vertex() As D3DVECTOR
```

فأما fCount فيمثل عدد الوجوه , وأما vCount فيمثل عدد الرؤوس , وأما L و L2 فسنستخدمهما في الحلقات التكرارية , والآن نعلن عن رأس مؤقت يستخدم في قراءة الرؤوس :

```
Dim Vec As D3DVECTOR
```

بعد هذا نضع عدد وجوه المجسم في المتغير الخاص بعدد الوجوه :

```
fCount = rMesh.GetFaceCount
```

ومادمنا نعرف أن كل وجه هو في الواقع مثلث , فإن عدد الرؤوس سيكون ٢ , لأن العد يبدأ من صفر , فتكون لدينا ثلاث نقاط , ٠ و ١ و ٢ :

```
vCount = 2
```

نعيد تعريف المصفوفات , بناء على عدد الوجوه , وعدد الرؤوس :

```
ReDim Face(fCount)
ReDim Vertex(fCount, 3)
```

بعد هذا نبدأ الحلقة التكرارية الأولى :

```
For L = 0 To fCount – 1
```

نستخدم Face في جلب بيانات الوجه من الـ Mesh :

```
Set Face(L) = rMesh.GetFace(L)
```

نبدأ الحلقة التكرارية الثانية :

```
For L2 = 0 To vCount
```

نجلب بيانات النقط (الرؤوس) ونضعها في عناصر مصفوفة الرؤوس :

```
Face(L).GetVertex L2, Vertex(L, L2), Vec
```

نغلق الحلقة التكرارية الثانية :

```
Next L2
```

نغلق الحلقة التكرارية الأولى :

```
Next L
```

وهكذا نكون قد قرأنا المجسم بالكامل , ونحن لا يهمنا الوجوه من حيث ترتيب الأماكن , ولكن الرؤوس فقط تهمنا .

إنشاء الوجوه

قبل أن نبدأ , لقد أنشأت وحدة برمجية Module لتساعدنا على إنشاء كائنات Direct3D RM Mode بسهولة , مثل التي أنشأتها في Direct 3D IM Mode , وهي كالتالي :

أولا الإعلانات العامة :

```
Public Dx As New DirectX7
Public Dd As DirectDraw4
Public Primary As DirectDrawSurface4
Public Back_Buffer As DirectDrawSurface4
Public Ddsd As DDSURFACEDESC2

Public D3d As Direct3DRM3
Public Device As Direct3DRMDevice3
Public ViewPort As Direct3DRMViewport2
Dim Light As Direct3DRMLight
Dim Shadow As Direct3DRMLight

Public WorldFrame As Direct3DRMFrame3
Public CameraFrame As Direct3DRMFrame3
```

```
Public LightFrame As Direct3DRMFrame3


Public Const PI  As Single = 22 / 7
Public Const RAD As Single = PI / 180
```

بعد هذا نضع دالة إنشاء Direct3D كالتالي :

```
Sub CreateD3DRM(Width As Long, Height As Long, Bpp As
Long)
```

ثم نضع تعريفات Direct Draw :

```
Set Dd = Dx.DirectDraw4Create("")
Dd.SetCooperativeLevel Form1.hWnd, DDSCL_ALLOWREBOOT
Or DDSCL_EXCLUSIVE Or DDSCL_FULLSCREEN
Dd.SetDisplayMode Width, Height, 16, 0, DDSDM_DEFAULT


Ddsd.lFlags = DDSD_CAPS Or DDSD_BACKBUFFERCOUNT
Ddsd.ddsCaps.lCaps = DDSCAPS_3DDEVICE Or
DDSCAPS_COMPLEX Or DDSCAPS_FLIP Or
DDSCAPS_PRIMARYSURFACE
```

```
Ddsd.lBackBufferCount = 1


Set Primary = Dd.CreateSurface(Ddsd)

Ddsd.ddsCaps.lCaps = DDSCAPS_BACKBUFFER

Set Back_Buffer =

Primary.GetAttachedSurface(Ddsd.ddsCaps)
```

ثم إنشاء كائن Direct3D :

```
Set D3d = Dx.Direct3DRMCreate
```

ثم إنشاء السواقة :

```
Set Device =

D3d.CreateDeviceFromSurface("IID_IDirect3DHALDevice",

Dd, Back_Buffer, D3DRMDEVICE_DEFAULT)
```

ثم نضع خيارات التعامل مع السواقة :

```
Device.SetBufferCount 2
```

```
Device.SetQuality D3DRMLIGHT_ON Or
D3DRMRENDER_GOURAUD
Device.SetTextureQuality D3DRMTEXTURE_NEAREST
Device.SetRenderMode
D3DRMRENDERMODE_BLENDEDTRANSPARENCY
```

ثم ننشئ الفرامات الرئيسية :

```
Set WorldFrame = D3d.CreateFrame(Nothing)
Set CameraFrame = D3d.CreateFrame(WorldFrame)
Set LightFrame = D3d.CreateFrame(WorldFrame)
```

ثم ننشئ الإضاءة , والظل :

```
Set Light = D3d.CreateLightRGB(D3DRMLIGHT_AMBIENT, 1,
1, 1)
Set Shadow = D3d.CreateLightRGB(D3DRMLIGHT_POINT, 2,
2, 2)
```

ثم نضيفهما إلى فرام الضوء :

```
LightFrame.AddLight Light

LightFrame.AddLight Shadow
```

وأخيرا ننشئ منفذ الرؤية :

```
Set ViewPort = D3d.CreateViewport(Device,

CameraFrame, 0, 0, Width, Height)
```

هذا عن دالة إنشاء DirectX , ولكنني أضفت دالة أخرى , تستخدم عند انتهاء البرنامج , لتقوم بمسح المتغيرات الرئيسية من الذاكرة , وهي كالتالي :

```
Sub Unload_App()

On Error Resume Next

Set Dx = Nothing

Set Dd = Nothing

Set Primary = Nothing

Set Back_Buffer = Nothing


Set D3d = Nothing
```

```
Set Device = Nothing

Set WorldFrame = Nothing

Set CameraFrame = Nothing

Set LightFrame = Nothing

Set ViewPort = Nothing

End

End Sub
```

الآن , تعال نبدأ العمل , أنشئ مشروعا جديدا , وأضف إليه وحدة برمجية , مثل التي شرحناها منذ قليل , ثم اذهب إلى كود الفورم , وفي قسم الإعلان العام , General Declaration , أعلن عن المتغيرات الأساسية كالتالي :

```
Dim Frame As Direct3DRMFrame3

Dim Mesh As Direct3DRMMeshBuilder3

Dim Face As Direct3DRMFace2
```

فأما Frame فهو الذي سيتم التحكم من خلاله , في مكان , ودوران , وحجم المجسم , وأما Mesh فهو المجسم الذي سيتم اضافة الوجه إليه , وأما Face فهو الوجه الذي سننشئه .

ثم نذهب إلى إجراء Form_Load , وفيه ننشئ كائنات DirectX كالتالي :

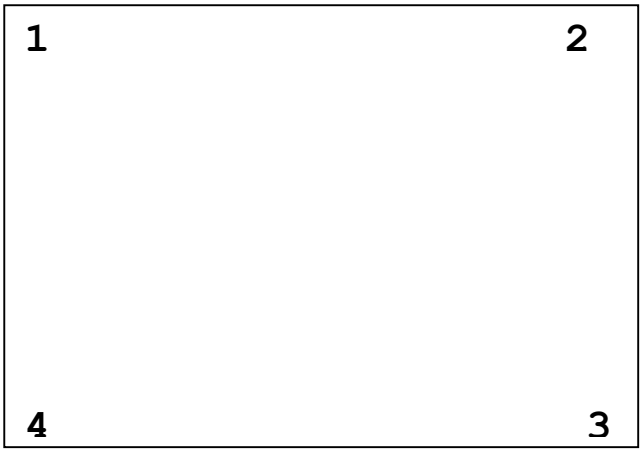
```
CreateD3DRM 800, 600, 0
```

بعد هذا نأمر Direct3D بإنشاء الوجه :

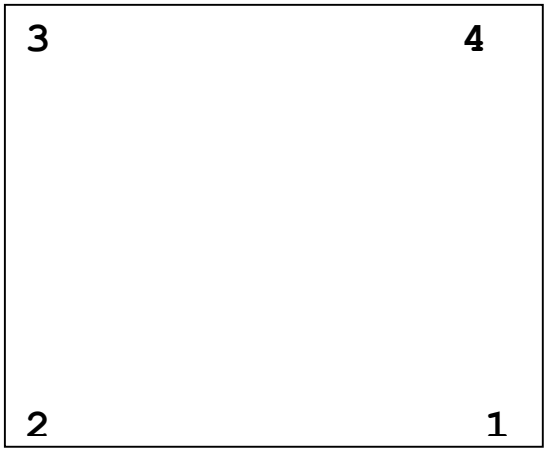
```
Set Face = D3d.CreateFace()
```

والآن حان الوقت لرسم الوجه عن طريق أربعة رؤوس , وقبل أن أكتب كود رسم كل رأس من الرؤوس , سأشرح لك طريقة الرسم .

رسم الوجه يكون بترتيب الرؤوس , على شكل مربع (شكل رباعي) , وترتيب رؤوسه يكون باتجاه دوران عقارب الساعة كالتالي :



أو كالتالي :



وليس من المهم أين تكون أول نقطة يتم إنشاءها من الوجه , ولكن المهم أن يكون ترتيب النقط باتجاه دوران عقارب الساعة .

ننشئ الوجه :

```
Face.AddVertex -2, 2, 0

Face.AddVertex 2, 2, 0

Face.AddVertex 2, -2, 0

Face.AddVertex -2, -2, 0
```

بعد هذا نغير لون الوجه , ولو لم نغير لون الوجه لصار تلقائيا باللون الأبيض :

```
Face.SetColorRGB 1, 0, 1
```

لاحظ ترتيب رؤوس ... ثم بعد هذا نأمر Direct3D بإنشاء المجسم :

```
Set Mesh = D3d.CreateMeshBuilder()
```

بعد هذا نضيف الوجه إلى المجسم :

```
Mesh.AddFace Face
```

بعد هذا ننشئ الفرام الذي سيحتوي على المجسم :

```
Set Frame = D3d.CreateFrame(WorldFrame)
```

بعد هذا نضيف المجسم إلى الفرام :

```
Frame.AddVisual Mesh
```

هذا كل شيء , والآن نبدأ في إظهار المجسم , كما تعلمنا في الدروس السابقة , ونحن نعرف أن الوجه يظهر من اتجاه واحد فقط , أي أنك تراه من أمامه فقط , أما لو كانت الكاميرا خلف الوجه فلن يظهر شيء .

وقد خصصت المثال الخامس والعشرون , من مجلد الأمثلة , لشرح كيفية تكوين الوجوه , ورسمها , و تضمينها مع المجسمات , ثم إظهارها على الشاشة .

الإكساء

هناك طرق عديدة للإكساء , أفضلها , وأكثرها ضمانا , هي عملية إكساء المجسم عن طريق إكساء وجوهه , وجه , وجه , وقد تعلمنا كيف نقرأ المجسم وجها وجها , وسنتعلم من خلال الدروس القادمة , كيف نعيد إنشاؤه مرة أخرى , بعد إجراء عمليات التلوين , والإكساء على الوجوه , ولكن أولا علينا أن نبدأ بمعرفة طريقة إكساء الوجه الواحد .

تعال معا نعد للمثال السابق , والذي تعلمنا من خلاله كيفية إنشاء وجه , ونأخذ نسخة منه , لنتعلم فيها كيف نكسوا هذا الوجه .

أولا أعلن عن متغير جديد , للإكساء , كالتالي :

```
Dim Texture As Direct3DRMTexture3
```

ثم تأمر Direct3D بتحميل هذا ملف الصورة على هذا المتغير :

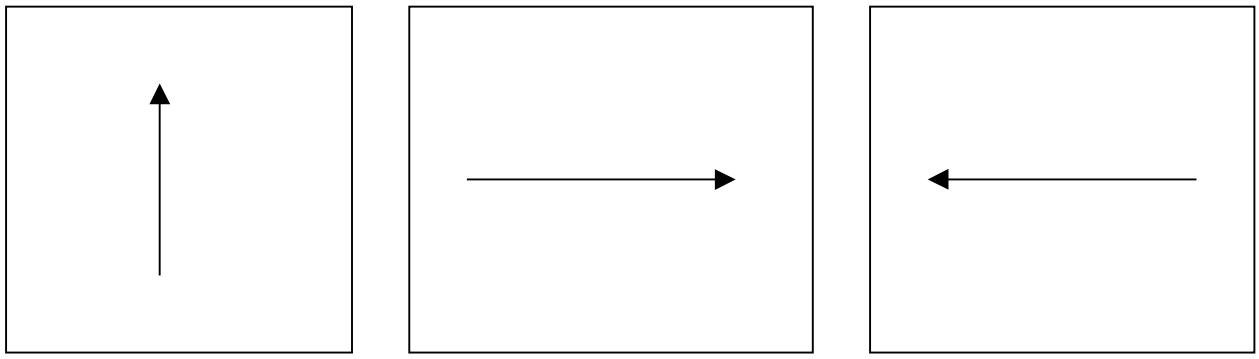
```
Set Texture = D3d.LoadTexture(App.Path &
"\texture.bmp")
```

بعد هذا نأمر نكسوا الوجه بالإكساء الذي صنعناه :

```
Face.SetTexture Texture
```

هكذا تم وضع الإكساء على الوجه , ولكنه لن يظهر لو شغلنا البرنامج , وذلك لأننا لم نقم بإعداد Texture Coordinates , فما هي Texture Coordinates ؟؟

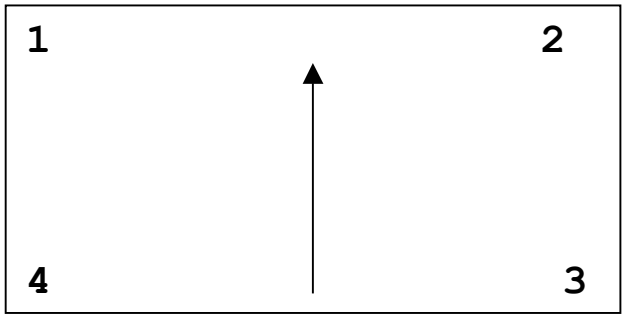
تقوم Texture Coordinates بعدد من الأشياء , مثل تحديد اتجاه الإكساء , كما ترى :



يشير السهم في كل وجه من الوجوه السابقة إلى اتجاه الإكساء , كما تقوم Texture Coordinates بتحديد عدد مرات تكرار الإكساء على الوجه الواحد , ولتفهم معي معنى هذا , تخيل معي وجه كبير جدا عبارة عن حائط مثلا , ولدينا إكساء عبارة عن قالب طوب , ونحن نريد إكساء هذا الحائط , بهذا الإكساء , لذا فسنقوم بتكرار الإكساء عدد كبير من المرات على الحائط , حتى يبدو كأنه مبني بهذا الطوب .

وحتى لو لم تكن تريد أن تغير اتجاه الإكساء , أو عدد مرات تكراره , فأنت بحاجة لتحديد Texture Coordinates حسب رغبتك هذه .

لنفترض أننا نريد أن نرسم Texture بدون أي تأثيرات , كالتالي :



وأننا نريد أن يكون عدد مرات الإكساء هنا هو مرة واحدة فقط , ونحن نحدد Texture Coordinates عن طريق ثلاثة معاملات , الأول هو رقم الرأس , والثاني هو قيمة X والثالث هو قيمة Y , فسيكون إنشاء أول نقطة كالتالي :

```
0 , 0 , 0
```

وأما النقطة الثانية فستكون :

```
1 , 1 , 0
```

وأما النقطة الثالثة فستكون :

```
2 , 1 , 1
```

وأما النقطة الرابعة فستكون :

```
3 , 0 , 1
```

وهكذا تكون معاملات Texture Coordinates على الترتيب هي – من اليسر إلى اليمين - :

رقم النقطة – قيمة X - قيمة Y .

وهكذا سننشئ Texture Coordinates الخاصة بالوجه , ليكون اتجاه Texture طبيعي , هكذا :

```
Face.SetTextureCoordinates 0, 0, 0
Face.SetTextureCoordinates 1, 1, 0
Face.SetTextureCoordinates 2, 1, 1
Face.SetTextureCoordinates 3, 0, 1
```

ولو كنا نريد أن نكرر الإكساء أكثر من مرة , فعلينا تغيير أرقام U و V أو , كما تعلمنا X و Y .

الآن لو فمت بتشغيل المثال فسيظهر الإكساء الذي صنعته .

وقد خصصت المثال السادس والعشرون , من مجلد الأمثلة لعمل الإكساء على الوجوه .

أمثلة على اتجاهات الإكساء

ربما تزال Texture Coordinates غير مفهومة بالنسبة لك , ولذا سأضع لك أربع أمثلة , على الحالات الأربع لاتجاه الإكساء في Direct3D RM Mode :

أولا / الاتجاه الطبيعي للصورة :


```
Face.SetTextureCoordinates 0, 0, 0
Face.SetTextureCoordinates 1, 1, 0
Face.SetTextureCoordinates 2, 1, 1
Face.SetTextureCoordinates 3, 0, 1
```

ثانيا / الصورة مائلة :

```
Face.SetTextureCoordinates 0, 1, 0

Face.SetTextureCoordinates 1, 1, 1

Face.SetTextureCoordinates 2, 0, 1

Face.SetTextureCoordinates 3, 0, 0
```

ثالثا / الصورة مقلوبة :

```
Face.SetTextureCoordinates 0, 1, 0

Face.SetTextureCoordinates 1, 0, 0

Face.SetTextureCoordinates 2, 0, 1

Face.SetTextureCoordinates 3, 1, 1
```

رابعا , الصورة مائلة بالاتجاه الآخر :

```
Face.SetTextureCoordinates 0, 0, 0

Face.SetTextureCoordinates 1, 0, 1

Face.SetTextureCoordinates 2, 1, 1

Face.SetTextureCoordinates 3, 1, 0
```

تكوين المجسمات

يتكون المجسم كما قلنا من عدة وجوه , فبرسم عدد من الوجوه , وتضمينها كلها في Mesh واحد , وإضافة هذا الـ Mesh إلى فرام , فنحن قد كونا مجسم , فمثلا لرسم صندوق نرسم وجوهه الستة كما تعلمنا , ثم نضيفها جميعا إلى الميش , ولكننا عندنا فكرة مسبقة عن هذا بالفعل , لذا لن أضع عنه أي أمثلة , وتعال لنفهم ما قصدته عندما قلت (تكوين المجسمات) .

تعلمنا فيما سبق (قراءة المجسمات) , وتحويل المجسم إلى عدد من الأوجه , والنقاط , ونحن هنا لنعيد تركيب المجسم , عن طريق أوجهه , ونقاطه , بعد أن نجري العمليات على هذه الأوجه .

لنبدأ العمل , أنشئ مشروعا جديدا , وأضف إليه الوحدة البرمجية التي نستخدمها في إنشاء كائنات Direct3DRM , ثم اذهب إلى كود الفورم , وأعلن عن المتغيرات الرئيسية كالتالي :

```
Dim Frame As Direct3DRMFrame3

Dim Mesh As Direct3DRMMeshBuilder3

Dim Face() As Direct3DRMFace2

Dim Vertex() As D3DVECTOR

Dim fCount As Long
```

```
Dim L As Long, L2 As Long
```

فأما Frame فهو الفرام الذي سيقوم بالتحكم في المجسم , وأما Mesh فهو الذي سنقوم بقراءته وتحويله إلى عدد من الوجوه , والنقط , ثم سنعيد كتابته من جديد , وأما Face() فهي مصفوفة الوجوه التي سنقرأها , وأنا Vertex() فهي مصفوفة النقط التي سنقرأها , وأما fCount فهو متغير سنضع فيه عدد الوحوه المتوفرة في Mesh , وأما L و L2 فهما متغيرين رقميين سنستخدمهما في الحلقات التكرارية .

بعد هذا نذهب إلى حدث إجراء تحميل النموذج , وفيه ننشئ كائنات دايركت إكس من الوحدة البرمجية كالتالي :

```
CreateD3DRM 800, 600, 0
```

بعد هذا ننشئ المجسم :

```
Set Mesh = D3d.CreateMeshBuilder
```

ثم نقوم بتحميل المجسم من الملف X في Mesh :

```
Mesh.LoadFromFile App.Path & "\pyramid.x", 0, 0,
Nothing, Nothing
```

ثم نقوم بنداء دالة قراءة المجسم وهي كالتالي :

```
Sub ReadMesh()
fCount = Mesh.GetFaceCount
Dim D As D3DVECTOR
ReDim Face(fCount - 1)
ReDim Vertex(fCount - 1, 2)
For L = 0 To fCount - 1
Set Face(L) = Mesh.GetFace(L)
For L2 = 0 To 2
Face(L).GetVertex L2, Vertex(L, L2), D
Next L2
Next L
End Sub
```

أما دالة إعادة كتابة المجسم فهي كالتالي :

```
Sub WriteMesh()

Dim Tex As Direct3DRMTexture3

Set Tex = D3d.LoadTexture(App.Path & "\texture.bmp")

Set Mesh = Nothing

Set Mesh = D3d.CreateMeshBuilder

For L = 0 To fCount - 1

Set Face(L) = Nothing

Set Face(L) = D3d.CreateFace

For L2 = 0 To 2

Face(L).AddVertex Vertex(L, L2).x, Vertex(L, L2).y,

Vertex(L, L2).z

Next L2

Face(L).SetTexture Tex

Face(L).SetTextureCoordinates 0, 0, 0

Face(L).SetTextureCoordinates 1, 0, 4

Face(L).SetTextureCoordinates 2, 4, 4

'Face(L).SetTextureCoordinates 3, 1, 0

Mesh.AddFace Face(L)

Next L

For L = 0 To fCount - 1
```

```
Set Face(L) = Nothing
Next
Set Tex = Nothing
End Sub
```

نعود إلى إجراء Form_Load , ونكتب أمر قراءة , وكتابة المجسم :

```
ReadMesh
WriteMesh
```

حسنا .. المجسم الآن جاهز للوضع في Frame ثم رؤيته على الشاشة , وقد وضعت المثال السابع والعشرين , من مجلد الأمثلة , بهدف بيان قراءة وكتابة مجسم .

وهذا هو هدفنا من محاولة دمج كلا من Direct3dIM و Direct3D RM , وهو قراءة المجسمات , وتكوينها باستخدام Direct3D RM Mode ثم استخدامها ورسمها , في Direct3D IM Mode .

المجسمات المتحركة Animation Meshes.

معظم برامج العناصر ثلاثية الأبعاد , مثل 3D Studio Max تمنح مستخدم البرنامج القدرة على صنع المجسمات ثلاثية الأبعاد , بالإضافة إلى تحريكها , وحفظ هذا التحريك في مجموعة من اللقطات , كل لقطة تسمى Frame , والمشهد العام يسمى Scene , وعند ترتيب هذه الـ Frames في الـ scene وعرضها متتابعة بسرعة معينه , فهي تظهر بالنسبة للمستخدم كأن المجسم يتحرك بالفعل , وليس عبارة عن مجموعة لقطات ثابتة .

ولما كان هذا التحريك يعطي مظهرا جماليا , وواقعية على المجسم , فقد لزم على DirectX أن توفر لك إظهار المجسمات المتحركة .

عند تشغيل برنامج 3D Studio Max , وبعد أن تقوم بإنشاء مجسمك , وتحريكه , تقوم بتخزينه عن طريق Export , الذي سيقوم بتخزين مجسمك على هيئة File.3DS , ثم تستخدم أحد برامج التحويل مثل DeepExploration , والذي سيقوم بتحويل مثالك إلى File.X .

بعد هذا تنشئ مشرعا جديدا باستخدام Visual Basic , ثم اضف له الوحدة البرمجية التي نستخدمها في انشاء Direct3D , ثم اذهب إلى كود الفورم , وفيه أعلن عن المتغيرات التالية :

```
Dim Anim As Direct3DRMAnimationSet2
Dim Frame As Direct3DRMFrame3
```

```
Dim Time As Single
```

أما Anim فهو الكائن الذي سينوب عوضا عن Mesh , وهو يستخدم في تحميل الملفات التي تحتوي على ANIMATION DATA شرط أن تكون من النوع X.* , وأما Time فسنستخدمه في تحديد الفرام الذي يظهر على الشاشة , وأما Frame فهو الكائن الذي سيظهر لنا Anim على الشاشة , وكذلك يتحكم فيه , كما يتحكم في Mesh .

الآن نذهب إلى حدث تحميل الفورم , وفيه ننشئ أولا كائن Anim و كائن Frame , كالتالي :

```
Set Anim = D3d.CreateAnimationSet()
Set Frame = D3d.CreateFrame(WorldFrame)
```

بعد هذا نأمر كائن Anim بأن يحمل بيانات المجسم , والتحريك , من الملف الخارجي :

```
Anim.LoadFromFile App.Path & "\AnimMan.x", 0, 0,
Nothing, Nothing, Frame
```

لاحظ أن آخر متغير من متغيرات الدالة السابقة هو Frame , وهكذا يعتبر Frame هو المتحكم في Anim , ثم نحدد الفرام الحالي من فرامات تحريك المتغير Anim كالتالي :

```
Anim.SetTime 0
```

وصفر هو أول فرام من فرامات تحريك المجسم , والمجسم الذي أستخدمه في مثالي مكون من ٢٠ فرام , لذا فإن تحريكه , من ٠ إلى ١٩ , كالتالي :

```
Time = Time + 0.2

If Time > 19 Then Time = 0

If Time Mod 1 = 0 Then Anim.SetTime Time
```

والآن نعرف كيف ننشئ مجسم متحرك , وكيف نضيفه إلى الكود , وقد اضفت مثالا من مجلد الأمثلة , يسمى المثال الثامن والعشرون , على التحريك Animation :

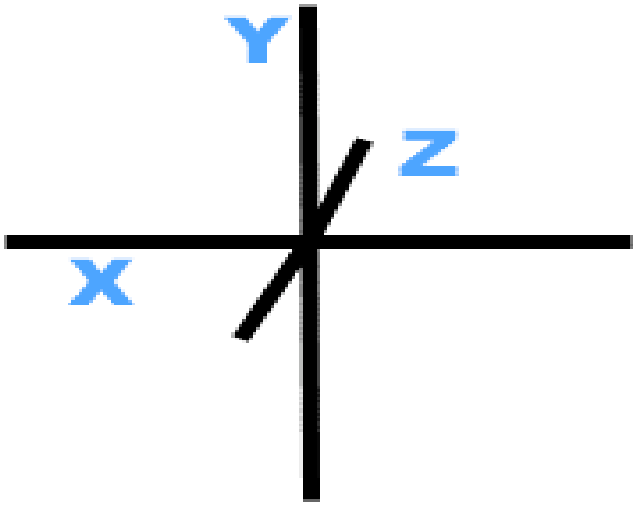
ملاحظة : التحريك Animation في DirectX يعتمد على ثلاثة أشياء فقط , وهي المكان , والدوران , والحجم (Position – Rotaion – Scale) , ولا يعمل مع الإجراءات الأخرى التي يوفرها لك Max مثل Bind أو Taper أو غير ذلك .

أسس التحريك

نحن عندما ننشئ برنامج , أو لعبة ثلاثية الأبعاد , فأول ما ننشئه منها هو العالم , والعالم يجب أن يكون ثلاثي الأبعاد حتى يكون مطابقا للواقع , فنحن نعيش في عالم ثلاثي الأبعاد (رباعي الأبعاد في الواقع) , والعالم الذي ننشئه يتكون من ثلاثة محاور , وهي X و Y و Z , ويتم تحديد اتجاهات هذه المحاور طبقا للكاميرا التي ترى العالم , والتي هي بمثابة عين الإنسان .

فالكاميرا الطبيعية ستؤسس العالم على أن X و Y هما بعدي الشاشة , أما Z فهو البعد الثالث , وهو عمودي على سطح الشاشة , يتجه للداخل , أو للخارج , كهذا :

الصورة السابقة توضح لك فكرة العالم ثلاثي الأبعاد , ولكنها لا توضح فكرة المحاور Z , Y , X , ولكن هذه الصورة تفعل :



فأنت ترى من هذه الصورة , أن X و Y هما بعدي سطح الشاشة , أما Z فهو ما يسمى بالعمق , ولكن هذا بالنسبة للعالم فحسب , أما بالنسبة للعناصر التي بداخل العالم , فلكل عنصر محاوره الخاصة به , ولتفهم ما أقول تعال نضرب مثال .

لنفترض أنك الآن تقف في حجرة من بيتك , أي حجرة , وتقف بحيث يكون وجهك باتجاه النافذة , في هذه الحالة سيكون المحاور X و Y هما يحددان مكان عينك في الحجرة , أما المحور Z فهو يتجه باتجاه النافذة , لنفترض أنك درت باتجاه أحد الحوائط بحيث أصبح المحور Z بالنسبة لك متجه نحو الحائط , في حين أن X يتجه نحو النافذة , و Y كما هو , هكذا أصبح لك محاورك الخاصة التي تخالف محاور الحجرة , اتلي بها المحور Z يتجه إلى النافذة .

وهكذا لو تحركت للأمام فأنت تتحرك على المحور Z الخاص بك , ولكن لو تحركت على المحور Z الخاص بالحجرة فأنت ستتحرك بجانبك .

هكذا التحريك في Frames , فإذا أنت كتبت :

```
Frame.SetPosition NoThing , X , Y , Z
```

فأنت بهذا تحدد موقعك بالنسبة للعالم , وليس بالنسبة للفرام , وبالتالي فسيتحرك الفرام نحو النقطة التي رسمتها بالنسبة للعالم .

أما لو كنت تريد الفرام يتحرك للأمام بالنسبة إلى نفسه , فيمكنك أن تكتب التالي :

```
Frame.SetPosition Frame , X , Y , Z
```

فمثلا لو كانت Z تساوي واحد , فسيتحرك الفرام للأمام بمقدار نقطة واحدة , وهذا بغض النظر عن اتجاهه بالنسبة للفرام العالم , أو الأب .

الدمج بين نظامي Direct3D RM Mode

و

Direct3D IM Mode

لماذا نحتاج إلى الدمج بين النظامين , مادمنا نستطيع الاكتفاء بأي واحد منهما ؟؟

كما قلنا من قبل أن Direct3D IM هو أقوى بكثير من النظام الآخر , خاصة في صنع الوجوه والتعامل معها , كما يتميز بالسرعة عن النظام الآخر , وهو يتيح لك شيئين هامين جدا , أولهما هو القدرة على استخدام Matrix , والآخر هو إمكانية توجيه الكاميرا على عنصر معين , وتظل تنظر نحوه باستمرار , أو بمعنى آخر نستطيع عمل Direction للكاميرا .

ولكننا لا نستطيع الابتعاد تماما عن RM Mode لأنها يتيح لنا شيء هام جدا , وهو التعامل مع الملفات من نوع X .

ولذا فخطوات الدمج بين النوعين تتم على عدد من الخطوات :

١-إنشاء النوع Direct3D RM Mode .

٢-قراءة المجسم , وتخزين نقاطه في Vectors .

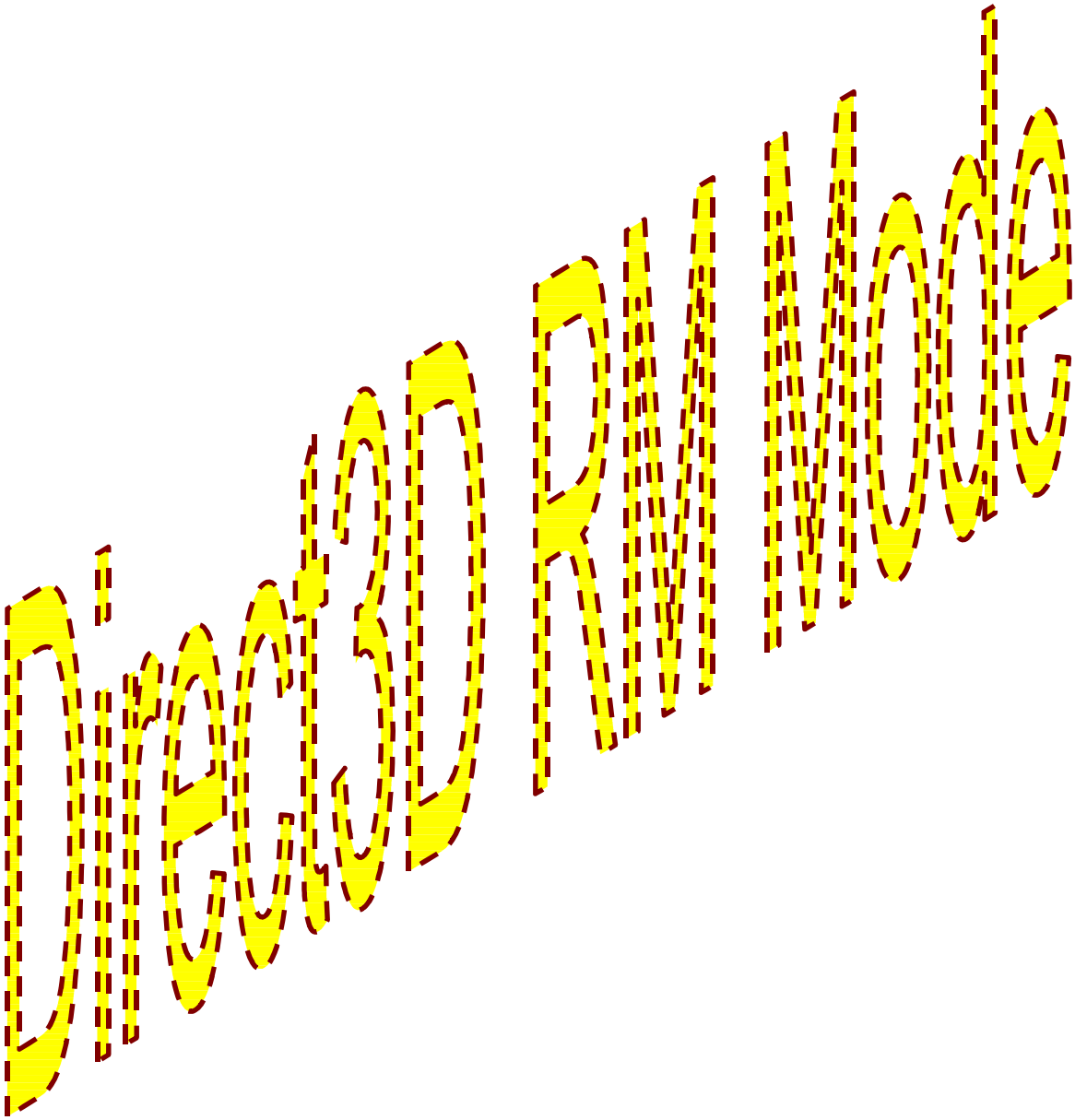
٣-حذف كائنات RM من الذاكرة .

٤-إنشاء النوع IM Mode .

٥-إنشاء المجسم مرة أخرى .

٦-عرضه في IM Mode .

وهكذا تستطيع (بطريقتك الخاصة) البرط بين النوعين Direct3D Retain Mode, و Direct3D Immediate Mode.



والآن , وبعد انتهاء الفصول التعليمية للكتاب , فكل ما أرجوه من اللّه , أن يفيد كل من يقرأه , وأن يكون مرجعا , شاملا , مفيدا , وممتعا , لكل من يقرأه , أو يقتني نسخه منه , وأن أكون قد وفقت في تقديم المادة العلمية التي فيه , وألا يكون هناك أي أجزاء غامضة , أو غير واضحة فيه .

الحقيقة أنني كنت أتمنى أن أزيد من عدد الصفحات , وعدد الأمثلة , والمعلومات المرفقة مع الكتاب , ولكن كانت هناك التزامات مع الشركة المنتجة هارين سوفت Harren Soft , وقد كنت أنوي كتابة الكتاب , ثم توزيعه مجانا على الشبكة , ولكن شاء اللّه سبحانه وتعالى أن ريى مدير الشركة نسخة من الكتاب (قبل أن أكمله) وقد أشار علي أنه يمكنه أن يقوم بعملية توزيع الكتاب , وهكذا انتقلت من مستوى الكتابة كهواية , إلى الكتابة كاحتراف , وأعترف أن هذا صعب جدا , فهناك إلتزامات , وجدول زمني أنت ملتزم به , وتليفونات كل نصف ساعة تسأل عن الجديد و ...

والشركة كانت تنزي وضع الكتاب على اسطوانات CD , مع عدد من الأدوات , والأمثلة المفيدة , وتوزيعه , ولكن هذا الـ CD كان سيتم توزيعه في مصر فقط , ولذا فقد رأيت أن يكون هناك نسختان , الأولى على CD ويتم توزيعها داخل مصر فقط , والأخرى يتم توزيعها على الإنترنت , ولما كانت الشركة ليست لها خبرات في عملية التوزيع على الإنترنت , فقد قررت توزيعه بنفسي على الإنترنت ...

أرجو من اللّه أن يكون قد وفقني , وأشكرك عزيزي القارئ على ثقتك الغالية , وأرجو من اللّه لك التوفيق دائما إن شاء اللّه