الرسم والتلوين باستخدام GDI+ و DirectX

لمبرمجي VB.NET

مقـــدمـــة

بسم الله والصلاة والسلام على رسول الله، وبعد:

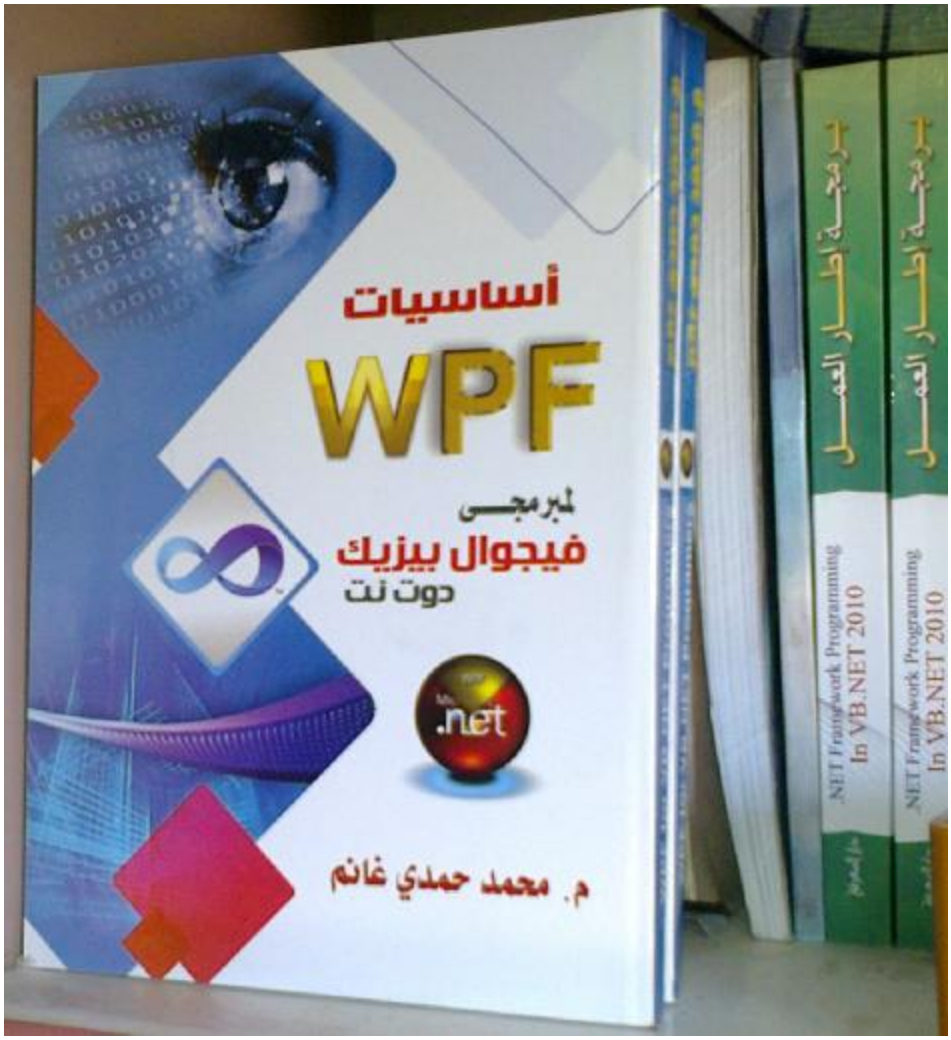
عندما كتبت مرجع "**من الصفر إلى الاحتراف فيجوال بيزيك دوت**" فـــي عـــام ٢٠٠٨، ومرجع "**من الصفر إلى الاحتراف برمجة نماذج الوزينـــدوز**" في عام ٢٠٠٩، نوهت فيهما إلى أنني أنوي كتابة مرجع عن برمجة الوسائط المتعددة في دوت نت، يتضمن الرسم والتلوين والصوت والفيديو والألعاب.. لكن هـــذا المرجع تأخر بسبب ترتيب أولوياتي في الكتابة عن مواضيع أخرى، ثم وجدت أنه لا ضرورة له، لأن **تقنية WPF** ــ التي تعتبر الجيل التالي لتقنية نمـــاذج الويندوز ــ قدمت نموذجا أسهل وأقوى للتعامل مع الرسوم والوسائط المتعددة والتحريك والمجسمات، كما أنها لا تعتمد على تقنية GDI+ المســـتخدمة فـــي نماذج الويندوز، بل تعتمد مباشرة على تقنية DirectX ممـــا يجعلهـــا أقـــوى وأسهل وتقدم إمكانيات رسومية غير مسبوقة.. لكل هذا كان من الطبيعـــي أن أتخذ قرارا بالكتابة عن الوسائط المتعددة من منظور WPF لا مـــن منظـــور نماذج الويندوز، لأواكب الأحدث.

ولكن لحسن الحظ، أني سبق أن ترجمت مرجع Mastering VB.NET إلى العربية ونشرته مجانا على الشبكة الدولية عام ٢٠٠٣، وهو يحتوي على جزء مفيد عن الرسم والتلوين والصور، كما أني أضفت إلى الترجمة مواضيع لـــم تكن في المرجع الأصلي، مثل التعامل مع الصور في الذاكرة مباشرة، وترجمة لمشروع تعليمي عن DirectX.. لهذا وجدت أنه من المناسب أن أجمع هـــذه الأجزاء في هذا الكتاب وأنشرها مجانا، لأعوض قراء كتبي عن المرجع الذي وعدتهم به ولم أكتبه، إلى أن أقدم بإذن الله الكتاب الجديد عن الوسائط المتعددة في WPF.

محمد حمدي غانم

٢٠١٤/٥/٢٥

كتب م. محمد حمدي غانم



- **<u>كتب لمبرمجي VB.NET</u>:**

١. أساسيات WPF لمبرمجي فيجوال بيزيك دوت نت
٢. المدخل العملي السريع إلى فيجيوال بيزيك دوت نت ٢٠١٠
٣. المبرمج الصغير: تعلم البرمجة بفيجيوال بيزيك دوت نت
٤. من الصفر إلى الاحتراف: فيجيوال بيزيك دوت نت ٢٠١٠
٥. من الصفر إلى الاحتراف: برمجة إطار العمل (VB.NET)

٦. من الصفر إلى الاحتراف: برمجة نماذج الويندوز (VB.NET)
٧. من الصفر إلى الاحتراف: برمجة قواعد البيانات (VB.NET)
٨. فيجيوال بيزيك وسي شارب: طريقك المختصر للانتقال من إحـــدى اللغتـــين إلى الأخرى

(وتوجد نسخة من كل كتاب لمبرمجي #C):

لتفاصيل أكثر عن هذه الكتب:

http://mhmdhmdy.blogspot.com/2010/09/blog-post_9555.html

- **كتب أدبية وفكرية:**

- خرافة داروين، حينما تتحول الصدفة إلى علم.. للتحميل:

http://mhmdhmdy.blogspot.com/2013/11/blog-post_29.html

- دلال الورد (ديوان شعر).. للتحميل:

http://www.mediafire.com/?n1qte7j9hdv1l98

- "حائرة في الحب (رواية).. للتحميل:

http://www.mediafire.com/?hd1jy6ca4ay3m9w

- **للتواصل مع الكاتب:**

- بريد الكتروني:

msvbnet@hotmail.com

- المدونة:

http://mhmdhmdy.blogspot.com

- قناة يوتيوب:

http://www.youtube.com/user/mhmdhmdy

- صفحة الشاعر محمد حمدي غانم:

https://www.facebook.com/Poet.Mhmd.Hmdy

- صفحة فيجوال بيزيك وسي شارب:

https://www.facebook.com/vbandcsharp

الفهرس

الفصل الأول

الرسم والتلوين باستخدام GDI+

الفصل الثاني

الأدوات الخاصة Custom Controls

الفصل الثالث

مشروع تعليمي بـــ Direct3D

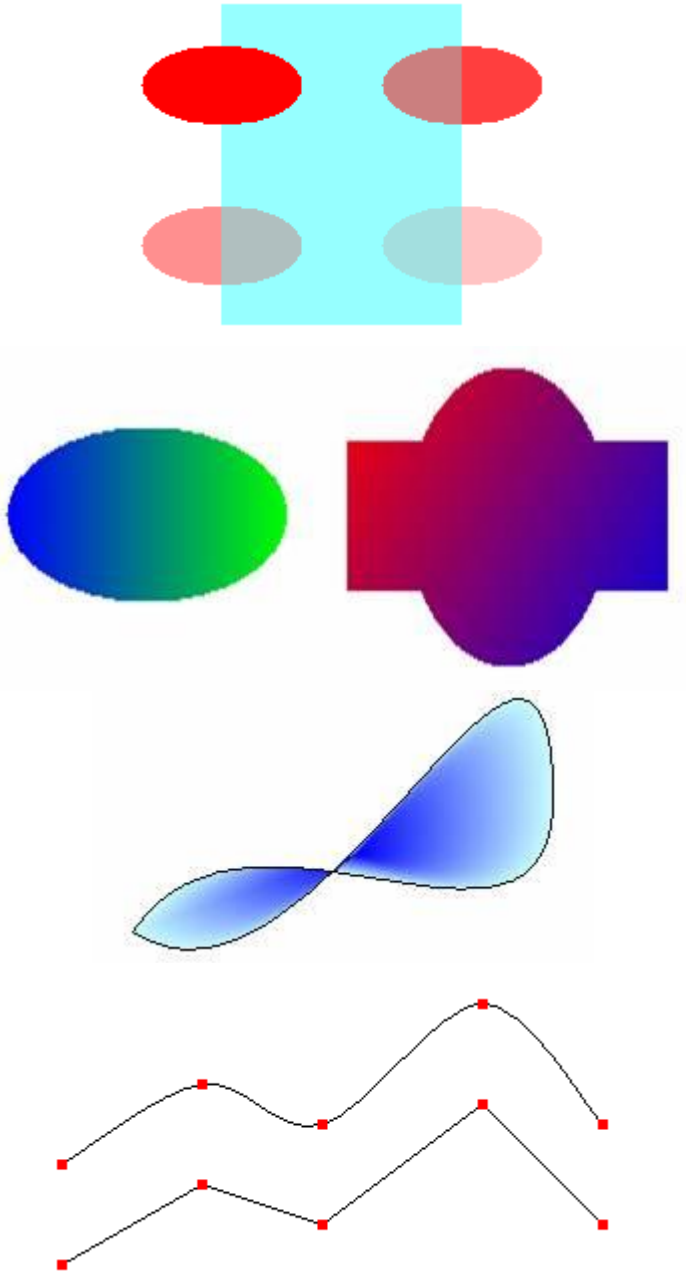
الفصل الأول

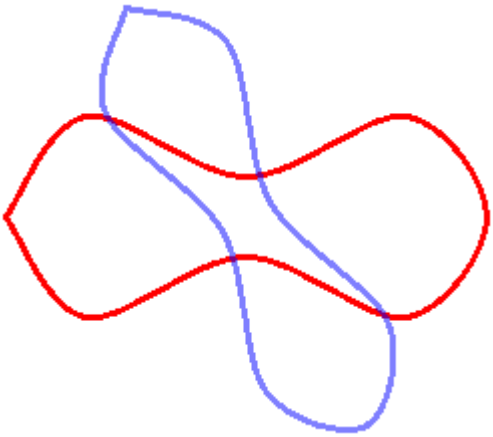
الرسم والتلوين باستخدام GDI+

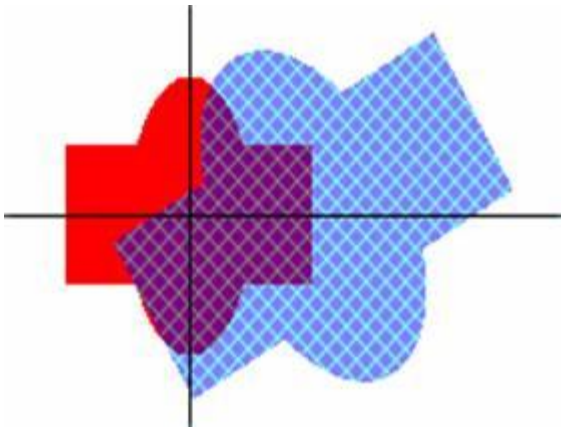
قدرات مدهشة:

هل ترى هذه الرسوم الرائعة؟

هل تصدّق أنّك تستطيع أن ترسم مثلها في كود VB؟







إنّ إطار العمال Net FrameWork. يمنحك العديد مـــن الفئـــات التـــي تستطيع أن تستخدمها في VB لإنشاء تطبيقات رسوم رائعة.

وتنقسم الرسومات Graphics إلى نوعيّن:

١- الرسوم الرياضيّة Vector: وهي التي ترســـمها باســـتخدام دوال VB، مثل دوال رسم الخطوط والمنحنيات التي تعتبر إحدى أنواع الرسوم الرياضيّة، مثل تلك التي رأيناها بأعلى.

٢- الصور Bitmaps: حيث يمكنك عرضها في العديد مـــن أدوات VB، كما يمكنك إجراء العمليّات عليها عن طريق معالجتها نقطة نقطة.

ولا غنى لك عن استخدام كلتا الوسيلتين، وإن كان استخدامك للصور سيكون أكثر تواترا.

ويمكنك الرسم في VB.NET، تقريبا على أيّ أداة تخطر ببالك، بما في ذلك مربّع النصّ والقائمة ListBox، وإن كان الأكثر شيوعا أن ترسم على النموذج ومربّع الصورة PictureBox.

ولمعظم الأدوات خاصيّة "الصورة" Image أو "صورة الخلفيّة" BackgroundImage، التي يمكنك تغيير قيمتها من نافذة الخصائص في وقت التصميم، أو بكتابة الكود الذي يغيّرهما في البرنامج.

ملاحظة:

عند وضع صورة في خلفيّة أيّ أداة أو نموذج في وقت التصميم، يتمّ نسخها من موقعها الأصليّ إلى ملفّات مشروعك، بحيث لا تقلق إذا تغيّر موقع برنامجك أو موقع الصورة الأصليّة.. أمّا لو استخدمت الكود في تحميل صورة، فإنّها تظلّ في مكانها الأصليّ.. لهذا يجب عليك تجنّب الأخطاء التي يمكن أن تحدث عند حذف هذه الصورة أو تغيّر موقعها.

التعامل مع الألوان

كائن اللون Color Object:

يمكنك تعريف متغيّر من هذا السجلّ كما يلي:

```
Dim myColor As Color
myColor = Color.Azure
```

إنّ سجلّ الألوان Color Structure يمنحك ١٢٨ لونا بأسمائها الإنجليزيّة (ولا أدّعي أنّني أعرف باللغة العربيّة هذا العدد من أسماء الألوان!!)، مثل اللون اللازورديّ (الأزرق السماويّ) Azure.. ويمكنك أن تتعرّف على هذه الألوان بمجرّد كتابة:

```
Color.
```

حيث ستظهر لك قائمة بكلّ قيم المرقّم Color.

ومن أهمّ الألوان اللون الشفّاف Transparent، حيث يمكنك أن تستخدمه لجعل خلفيّة بعض الأدوات ــ كاللافتة ــ شفّافة:

```
Lable1.BackColor = Color.Transparent
```

بالإضافة لهذا، فإنّ هذا السجلّ يمنحُك بعض الوسائل الهامّة، ومنها:

تكوين اللون من مركّباته FromARGB:

يمكنك أن تكوّن ملايين الألوان باستخدام هذه الوسيلة، حيث ترسل لها نسب الأحمر Red والأخضر Green والأزرق Blue لتعيد لك اللون المكوّن منها.. (طبعا اتضح لك أنّ الحروف RGB هي الحروف الأولى من أسماء الألوان الثلاثة).

وفي المثال التالي نغيّر لون خلفيّة مربّع النصّ باستخدام دالة تكوين اللون:

TextBox1.BackColor = Color.FromARGB(25,150,255)

وهنا يجب أن تلاحظ شيئين:

١- أنّ كلّ مكوّن من المكوّنات الثلاثة تنحصر قيمته بين ٠ و ٢٥٥، فإذا أرسلت للمعاملات أيّ عدد أكبر من هذا فسيكون تأثيره كتأثير العدد ٢٥٥.

٢- أنّ معكوس أيّ لون (نيجاتيف اللون) ينتج بطرح كلّ مكوّن من مكوّناته الثلاثة من ٢٥٥، وسيتّضح لك ذلك عندما تعرف أنّ مكونات اللون الأبيض هي (٢٥٥، ٢٥٥، ٢٥٥) ومكونات اللون الأسود هي (٠، ٠، ٠).

والجدول التالي يوضّح لك مكوّنات بعض الألوان الهامّة:

اللون	نسبة الأحمر R	نسبة الأخضر G	نسبة الأزرق B
الأسود	٠	٠	٠
الأزرق	٠	٠	٢٥٥
الأخضر	٠	٢٥٥	٠

السماوي	٠	٢٥٥	٢٥٥
الأحمر	٢٥٥	٠	٠
الأحمر البنفسجيّ	٢٥٥	٠	٢٥٥
الأصفر	٢٥٥	٢٥٥	٠
الأبيض	٢٥٥	٢٥٥	٢٥٥
الرمادي	١٢٨	١٢٨	١٢٨

وإذا أردت الحصول على لون فاتح من الألوان الماضية، فاستبدل العدد ١٢٨ بالعدد ٢٥٥ في كلّ مكوّنات اللون.

وهناك صيغة أخرى من هذه الوسيلة، تسمح لك بتحديد درجة شفافية اللون:

FromARGB(شفافية اللون ,نسبة الأحمر ,نسبة الأخضر ,نسبة الأزرق)

حيث تتراوح الشفافية ما بين ٠ (شفافية تامّة) و ٢٥٥ (إعتام تامّ).

ويسمّى معامل الشفافيّة "خليط ألفا" Alpha Blinding.. ولا أعرف تحديدا سبب وجود الحرف الأبجدي اللاتيني ألفا في هذا الاسم، وإن كنت أخمّن أنّه اسم الخوارزميّة Algorithm التي تقوم بخلط لون الكائن مع لون ما خلفه، لإعطاء إحساس الشفافيّة.. إنّ الكثير من الخوارزميّات البرمجيّة تحمل أسماء حروف لاتينية ومعاملات ومتغيّرات تدخل في كتابتها، كالخوارزميّة K-means مثلا.. والله أعلم.

ومن أروع استخدامات الشفافية، استخدامها لكتابة نصوص تبدو للرائي مجسّمة، وذلك برسم النصّ بلون ما (سنعرف كيفيّة ذلك لاحقا)، ثمّ رسم نفس النصّ بلون آخر وجعله شبه شفّاف، على أن يكون النصّ في المرّة

الثانية مزاحا قليلا رأسيّا وأفقيّا.. ويمكنك أن ترى هذه الطريقة في تطبيق TextEffects.

كما يمكنك استخدام الشفافية في رسم علامة مائيّة على الصورة عند نشرها على الإنترنت.. هذه العلامة المائيّة لن ترهق العين، ولكنّها ستعوق الآخرين عن استخدام هذه الصورة بدون تصريح منك.

والكود التالي يكتب جملة "MySite.com" على النموذج نصف شفّافة.

```
Dim F As New Font("Comic Sans MS", 20,
        FontStyle.Bold)
Dim B As New SolidBrush(Color.FromARGB(50, 230,
        80, 120))
Me.CreateGraphics.DrawString("MySite.com", F, B,
        100, 40)
```

تكوين لون معروف FromKnownColor:

تسمح لك هذه الوسيلة بتكوين اللون من قيم المرقّم KnownColor.. وهذه القيم تتدرج تحت طائفتين:

- ١٢٨ لونا بأسمائها الإنجليزيّة المعروفة، مثل KnownColor.Red .
- الألوان الخاصّة بنظام الويندوز (والتي يغيّرها المستخدم من خصائص سطح المكتب لتكون عامّة للويندوز)، مثل لون العناوين KnownColor.ActiveCaption، ولون نصوص العناوين KnownColor.ActiveCaptionText.. إلخ.

GetBrightness:

للحصول على درجة إضاءة اللون.

GetHue:

للحصول على درجة تدرّج اللون.

GetSaturation:

للحصول على درجة تشبّع اللون.

ToArgb:

استخدم هذه الوسيلة للحصول على العدد الدالّ على اللون (وهو يتكوّن من ٤ وحدات 4 Bytes)، حتّى تستطيع استخدامه مع التطبيقات الأخرى التي تتعامل مع الألوان كأرقام وليس ككائنات.. ومن أطرف استخدامات هذه الوسيلة، استخدامها لمقارنة لونين معا.. إنّ الجملة التالية مرفوضة برمجيّا:

```
If Me.BackColor = Color.Red Then
```

وذلك لأنّ اللون سجلّ Structure، ولا يمكن التأكّد من تساوي سجلين أو كائنين بعمليّة تساوي عاديّة "=".. في حالة الكائنات المرجعيّة نستخدم المعامل Is بدلا من العلامة "=".. ولكنّ السجلات كائنات قيميّة وليست مرجعيّة.. لهذا لا يمكن استخدام أيّ من المعاملين "=" أو Is.

فما العمل إذن؟

يمكنك استخدام الوسيلة Equals كالتالي:

```
If Me.BackColor.Equals(Color.Red) Then
```

أو يمكنك استخدام أيّ وسيلة تحوّل اللون من سجلّ إلى رقم أو نصّ، كالتالي:

```
If Me.BackColor.ToArgb = Color.Red.ToArgb Then
```

أو:

```
If Me.BackColor.ToString = Color.Red.ToString Then
```

الخصائص:

درجة شفافية اللون A ــ نسبة الأحمر R ــ نسبة الأخضــر G ــ نسبة الأزرق B.

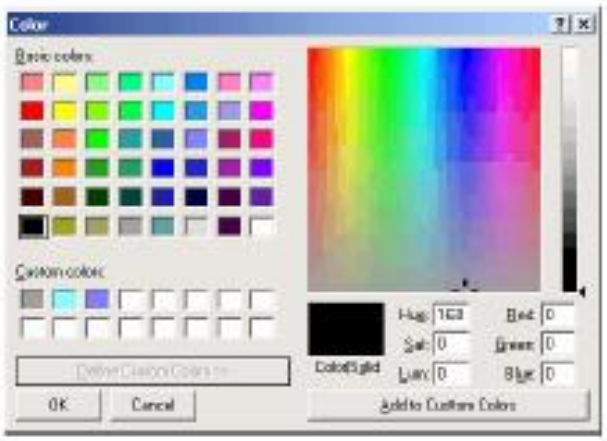
وأعتقد أنّها غنيّة عن التعريف.

والدالة التالية تعيد لك معكوس اللون الذي ترسله لها، مع المحافظة علــى درجة شفافيته كما هي:

```
Function RevColor(Clr As Color) As Color
    Return Color.FromARGB(Clr.A, 255 - Clr.R, _
        255 - Clr.G, 255 - Clr.B)
End Function
```

مربّع حوار اختيار اللون ColorDialog:

كثيرا ما تريد أن تمنح المستخدم الحرّيّة في اختيار ألوان خلفيّات النوافذ والأدوات.. في هذه الحالة لا بدّ أن تعرض مربّع حوار اختيار الألوان، ليختار منه المستخدم اللون الذي يناسبه.. ولحسن حظّك فإنّ VB يمنحك هذا المربّع جاهزا، وستجده في صندوق الأدوات باسم ColorDialog، حيث يمكنك أن تضيفه للنموذج كأيّ أداة أخرى.



تعال نرى خصائص مربّع الحوار هذا:

السماح بفتح كامل AllowFullOpen:

لمربّع حوار الألوان زرّ بعنوان "تعريف ألوان مخصّصة" Define Custom Colors.. لو جعلت هذه الخاصيّة False فسيكون هذا الزر غير فعّال ولن يتمكّن المستخدم من ضغطه، فإذا جعلت للخاصية AllowFullOpen القيمة True، فسيتمكّن المستخدم من ضغط هذا الزرّ، حيث سيظهر نموذج آخر، يستطيع المستخدم من خلاله تركيب أيّ لون يريده.

أيّ لون AnyColor:

لو جعلت هذه الخاصيّة True، فسيتمّ عرض جميع الألوان الأساسيّة المتاحة في مربّع حوار الألوان.

اللون Color:

هذه الخاصيّة تسمح لك باختيار اللون الذي سيكون محددا عندما يفتح المستخدم مربّع الحوار.. إنّك تحتاج لفعل هذا حتّى تعرض للمستخدم اللون الحالي للأداة التي سيغيّر لونها (لون خلفيّة مربّع النصّ مثلا)، أو لكي تعرض له آخر لون اختاره في المرّة السابقة.

كما أنّ هذه الخاصيّة تسمح لك بمعرفة اللون الذي اختاره المستخدم عند إغلاق مربّع الحوار.. انظر للمثال التالي:

```
' اللون الحاليّ لخلفيّة النموذج هو الذي سيكون محددا عند فتح مربع الحوار
ColorDialog1.Color = Me. BackColor
' عرض مربّع الحوار
' ويجب التأكّد من أنّ المستخدم قد ضغط موافق وليس إلغاء
If ColorDialog1.ShowDialog = DialogResult.OK Then
    ' في هذه الحالة نجعل خلفيّة النموذج باللون الذي اختاره المستخدم
    Me.BackColor = ColorDialog1.Color
End If
```

ألوان مخصّصة CustomColors:

في الجزء السفليّ من مربّع حوار الألوان منطقة يمكنك فيها وضع ١٦ لونا خاصّا بك.. تضعها كلّها أو بعضها.. ولكي تفعل ذلك، عرّف مصفوفة من الأعداد الصحيحة، وضع فيها الأرقام الدالة على الألوان التي تريد ظهورها في المنطقة المخصّصة، وضعها في هذه الخاصيّة:

```
Dim colors( ) As Integer = {222663, 35453, 7888}
ColorDialog1.CustomColors = colors
```

ونظرا لاستحالة حفظ أرقام الألوان، استخدم كائن الألوان لتعريف الألوان بأسماء واضحة، ثمّ حوّلها لأعداد صحيحة باستخدام الوسيلة ToARGB.. ونظرا لأنّ أوّل وحدة Byte على يسار اللون مخصّصة للشفافية، فإنّ ذلك قد يؤدّى لأرقام سالبة.. لهذا استخدم دالة العدد المطلق من فئة الدوالّ الرياضيّة Math.Abs للتأكّد من أنّ كلّ الأرقام موجبة:

```
Dim colors( ) As Integer =
    {Math.Abs(Color.Gray.ToARGB), _
     Math.Abs(Color.Navy.ToARGB),
     Math.Abs(Color.Teal.ToARGB)}
```

الألوان المتجانسة فقط SolidColorOnly:

اجعل هذه الخاصيّة True إذا كان برنامجك سيعمل على نظـــام يعـــرض ٢٥٦ لونا فقط.

وللتدريب على استخدام هذه الأداة، افحص الأمر Page Color والأمـــر Text Color في القائمة Customize في مشروع TxtPad في مجلّـــد برامج الفصل السادس.

رسم الأشكال الرياضيّة:

واجهة تصميم الرسومات:

Graphics Design Interface (GDI+):

إنّ GDI هي مجموعة من الفئات التي تمكّنك من إنشاء الرسوم والنصوص والصور، وهي بمثابة المحرّك الذي يستخدمه الويندوز نفسه في هذه العمليّات.. ولقد تطوّر GDI عبر الزمن، حتّى جاءت أحدث نسخة منه مع VS.Net تحمل اسم GDI+.

ونظرا لقوّة وثراء وسرعة أداء هذا المحرّك، فلقد ذهبت كل وسائل الرسم في VB6 بلا رجعة، ولم يعد بإمكانك إلا استخدام GDI+.

وتقع فئات هذا المحرّك في الفضاءات التالية، التي يجب عليك استيراد بعضها على الأقلّ قبل استخدامها في مشروعك:

System.Drawing
System.Drawing.Drawing2D
System.Drawing.Imaging
System.Drawing.Text

كيف ترسم:

لكي ترسم أيّ شيء يجب أن يتوافر لديك ما يلي:

سطح الرسم Surface:

يجب عليك قبل أن تبدأ الرسم، أن تختار السطح الذي سترسم عليه.. هذا السطح يمثّله كائن الرسوم Graphics object، الذي يمدّك بالوسائل التي ترسم الأشكال الأساسيّة وغير الأساسيّة.

وللرسم على نموذج أو أداة، يجب أن نتعامل مع كائن الرسوم الخاصّ بها، والذي تحصل عليه من الوسيلة "إنشاء رسوم" CreateGraphics.. كما يمكن الرسم على كائن صورة نقطيّة Bitmap Object، ووضع الصورة الناتجة بعد ذلك على النموذج أو الأداة، كما سنرى فيما بعد.

أداة الرسم:

لدينا أداتان رئيسيّتان نرسم بهما:

١. **القلم Pen**:

ويمكنك استخدامه في رسم الأشكال المكوّنة من خطوط، مثل الخطوط lines، والمستطيلات rectangles والمنحنيات curves.. وأهمّ خصائص القلم: لونه وسمك خطّه.

٢. **الفرشاة Brush**:

ويمكنك استخدامها في رسم المساحات الملوّنة.. وأهمّ خصائص الفرشاة: لونها والشكل الذي ستلوّنه وطريقة التلوين.

والمثال البسيط التالي يوضّح لك كيف ترسم خطّا على النموذج:

```
' تجهيز قلم أحمر سمك خطّه نقطتان
Dim redPen As Pen = New Pen(Color.Red, 2)
' نقطة بداية الخطّ
Dim point1 As Point = New Point(10,10)
' نقطة نهاية الخطّ
Dim point2 As Point = New Point(120,180)
' رسم الخطّ على النموذج
Me.CreateGraphics.DrawLine(redPen, point1, point2)
```

ويمكن دمج كلّ الخطوات الأربع السابقة في الخطوة الوحيدة التالية:

```
Me.CreateGraphics.DrawLine(
    New Pen(Color.Red, 2), _
    New Point(10,10), New Point(120,180))
```

وهناك صيغة أسهل للوسيلة DrawLine كالتالي:

```
Me.CreateGraphics.DrawLine(
    New Pen(Color.Red, 2), 10, 10, 120, 180)
```

حيث أرسلنا إحداثيات الخطّ منفردة، بدلا من إرسالها كنقطتي بداية ونهاية، وإنّ كانت الصيغة الأولى أوضح عند قراءتها من الصيغة الثانية.

كما يمكنك تعريف كائن رسوم ثمّ ربطه بكائن رسوم النموذج كالتالي:

```
Dim G As Graphics = Me.CreateGraphics
G.DrawLine(redPen, point1, point2)
```

أهمّ الكائنات المستخدمة في الرسم

تعتبر الكائنات التالية، كائنات أوّليّة في عمليّة الرسم:

كائن النقطة Point Object:

يمثّل هذه الكائن إحداثي نقطة: الإحداثيّ السينيّ X (موضع النقطة أفقيّا، بدئا من أقصى يسار الشاشة)، والإحداثيّ الصادي Y (موضع النقطة رأسيّا، بدئا من أعلى الشاشة).

ولإنشاء نقطة جديدة، يجب أن تحدد إحداثييها (الأفقيّ ثمّ الرأسيّ) كالتالي:

```
Dim P1 As New Point
P1.X = 34
P1.Y = 50
```

أو باختصار:

```
Dim P1 As Point
P1 = New Point(34, 50)
```

أو بشكل أكثر اختصارا:

```
Dim P1 As New Point(34 50)
```

ويقبل هذا الكائن الإحداثيات في صورة أعداد صحيحة Integer، فإذا أردت استخدام وحدة قياس غير الوحدة Pixel، فقد تكون قيم الإحداثيات أعدادا مفردة Single.. في هذه الحالة استخدم الكائن PointF، وهو مماثل لكائن النقطة Point، إلا إنّه يقبل الإحداثيات من النوع Single أو Double.. وبالمناسبة: الحرف F في اسم هذا الكائن هو اختصار لتعبير "علامة عشريّة" Floating-point.

كائن الحجم Size Object:

مشابه لكائن النقطة، ولكنّه يمتلك خاصيتين: العرض Width، والارتفاع Height، ويتمّ تعريفه كالتالي:

```
Dim S As Size
S.Width = 100
S.Height = 20
```

أو بطريقة أخرى:

```
Dim S As New Size(100,20)
```

وبطريقة ثالثة:

```
Dim S As New Size(New Point(100,20))
```

وهناك نوع آخر من كائن الحجم، هو كائن الحجم ذو الدقّة العشريّة SizeF، وهو مماثل لكائن الحجم Size، إلا إنّه يقبل معاملات من النوع Single و Double.

كائن المستطيل Rectangle Object:

يتطلّب تعريف المستطيل، موقع رأسه العلويّ الأيمن وعرضه وارتفاعه:

```
Dim box As New Rectangle(X, Y, العرض, الارتفاع)
```

وهناك صيغة أخرى باستخدام كائن النقطة وكائن الحجم:

```
Dim P As New Point(1,1)
Dim S As New Size(100,20)
Dim box As New Rectangle(P, S)
```

كما يمكنك أن تنشئ مستطيلا، ثمّ تضع قيم خصائصه كالتالي:

```
Dim box As New Rectangle( )
box.X = 1
box.Y = 1
```

```
box.Width = 100
box.Height = 20
```

وهناك نوع آخر من كائن المستطيل، هو كائن المستطيل ذو الدقّة العشريّة RectangleF، وهو مماثل للمستطيل Rectangle، إلا إنّه يقبل معاملات من النوع Single و Double.

كائن القلم Pen Object:

ذكرنا من قبل أنّ تعريف هذا الكائن يتطلّب اللون والسمك (ولو حذفت السمك من التعريف فسيتمّ اعتباره ١):

```
Dim P As Pen
P = New Pen(Color.Black, 3)
```

وهناك صيغة أخرى تسمح لك بتعريف اللون من كائن فرشاة كالتالي:

```
Dim patternPen As New Pen(brush, width)
```

وفي هذه الحالة يمكنك استخدام هذا القلم لتلوين الحدّ الخارجيّ لشكل بطريقة معيّنة (تحدّدها الفرشاة).

ولديك مجموعة من الأقلام الجاهزة محدّدة الألوان وسمكها ١، يمنحها لك مرقّم الأقلام Pens Structure.. مثل القلم الأزرق Pens.Blue.. ويمكنك أن تستخدم هذه الأقلام مباشرة بدون تعريف، كمعامل في أيّ وسيلة تتطلّب قلما.

خصائص القلم:

اتصّال الخط LineJoin:

تحدّد الطريقة التي ستلتحم بها الأشكال المتجاورة.

StartCap, EndCap:

تحددان شكل بداية ونهاية الخطّوط المرسومة.

DashCap

تحدّد شكل بداية ونهاية التقطيعات في الخطّ المتقطّع.

طراز التقطيع DashStyle:

استخدم هذه الخاصيّة لرسم الخطّ متقطّعا بالطراز الذي تريده.. وتأخذ هذه الخاصيّة قيمة من قيم المرقّم DashStyle وهي:

متّصل **Solid** ــ شــرطة **Dash** ــ مــنقّط **Dot** ــ شــرطة ونقطــة **DashDot** ــ شــرطة ونقطتــان **DashDotDot** ــ مخصّــص **Custom**.

نوع القلم PenType:

تحدّد نوع القلم، الذي هو أحد القيم التالية:

ملء مظلّل **HatchFilled** ــ لون متدرّج **LinearGradient** ــ لون متشعّب التدرّج **PathGradient** ــ لون ثابت **SolidColor** ــ مــلء بخامة **TextureFill**.

ستتساءل هنا، لماذا تتكلّم هذه القيم عن ملء الشكل باللون، مع أنّها خاصّة بالقلم وليس بالفرشاة؟
لا تنسَ أنّ القلم يرسم خطوطا، يمكن أن تكون سميكة.. ونحن نتكلّم عن كيفيّة ملء هذه الخطوط السميكة بالألوان.

كائن الفرشاة Brush Object:

تمكّنك الفرشاة Brush من ملء الأشكال بلون واحد أو بألوان متدرّجة أو بصورة.
ولا يمكنك تعريف فرشاة منفردة، فهذه الفئة ترثها أنواع الفرش المختلفة، وهي:

الفرشاة الصلبة SolidBrush:

تملأ الشكل بلون واحد.

Dim sBrush As New SolidBrush(لون الفرشاة)

وسنتعرّف عليها بالتفصيل لاحقا.

فرشاة التظليل HatchBrush:

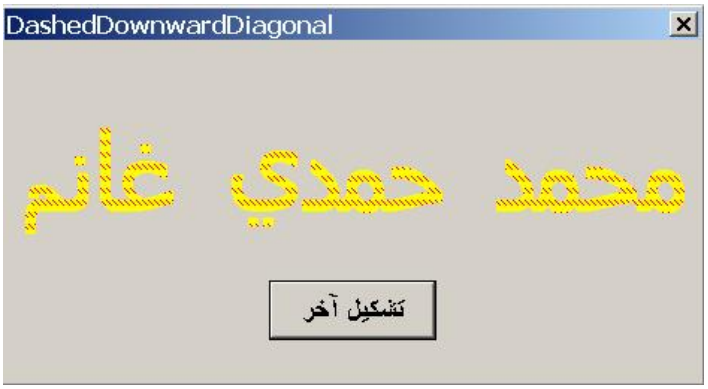
تملأ الشكل بتشكيلة معيّنة، من أكثر من ٥٠ تشكيلة جاهزة، يمكنك اختيارها من خلال المرقّم HatchStyle.. ويتمّ تعريف هذه الفرشاة كما يلي:

Dim hBrush As New HatchBrush (طراز الفرشاة,
لون الخلفيّة,لون خطوط التظليل)

وهذه بعض تشكيلات التظليل، التي يمكنك استخدامها كقيم للمعامل الأوّل:

ForwardDiagonal	خطوط قطريّة من أعلى اليسار إلى أسفل اليمين.
BackwardDiagonal	خطوط قطريّة من أعلى اليمين إلى أسفل اليسار.
Cross	خطوط أفقيّة ورأسيّة متقاطعة.
DiagonalCross	خطوط قطريّة متقاطعة.
Horizontal	خطوط أفقيّة.
Vertical	خطوط رأسيّة.

ويمكنك استعراض هذه التشكيلات المختلفة، في تطبيقي HatchBrushEnum و HatchBruchText في مجلّد برامج هذا الفصل.



الفرشاة المتدرّجة LinearGradientBrush:

تملأ الشكل بألوان متدرّجة.. ويبدأ التدرّج من لون ما وينتهي إلى لون آخر.

وسنتعرّف عليها بالتفصيل لاحقا.

فرشاة متشعّبة التدرّج PathGradientBrush:

تملأ مسار الشكل بألوان متشعّبة التدرّج، حيث يبدأ التدرّج بلون واحد، ولكنّه ينتهي إلى ألوان مختلفة.

وسنتعرّف عليها بالتفصيل لاحقا.

فرشاة الخامة TextureBrush:

تملأ الشكل بصورة.. ويتمّ تبليط Tiling الشكل بالصورة ليملأها كلّيا، بمعنى أنّ الصورة تتكرّر على حسب ما يتطلّب الأمر.. تخيّل مدى روعة الشكل الناتج من ملء دائرة أو منحنى بصورة معيّنة!

ملحوظة ١:

في أيّ نوع من أنواع الفرش السابقة (والأقلام ومسار الرسوم كذلك) يمكن استخدام معامل الشفافية في تكوين الألوان المرسلة كمعاملات لهذه الفئات، وبهذا يمكنك الحصول على أشكال شفافة بأيّ نسبة تريدها.. فمثلا: لو لوّنت أيّ شكل بالفرشاة التالية، فسيكتسب لونا أحمر نصف شفّاف، حيث سيمتزج لونه بلون خلفيّته:

```
Dim sBrush As SolidBrush
sBrush = New SolidBrush(Color.FromARGB(128, 255,
                0, 0))
```

ملحوظة٢:

يمنحك سجلّ الفرش Bruches Structure مجموعة من الفرش الجاهزة بألوان مختلفة.. مثل Brushes.Red التي تمنحك فرشاة تلوين حمراء.

كائن الرسوم Graphics Object:

أدوات كثيرة لا تتوقّعها تمنحك كائن رسوم، ممّا يمكّنك من الرسم عليها.. خذ مربّع النصّ مثلا!

ولكي تحصل على كائن الرسوم من الأداة، استخدم وسيلتها CreateGraphics.

ولكي تستخدم كائن الرسوم، استورد المكتبة Drawing2D في مشروعك أولا:

```
Imports System.Drawing.Drawing2D
```

بعد ذلك يمكنك تعريف متغيّر من هذا الكائن كالتالي:

```
Dim G As Graphics
```

ولكي ترسم بهذا الكائن على مربّع الصورة، استخدم الجملة التالية:

```
G = PictureBox1.CreateGraphics
```

ملحوظة:

يرتبط كائن الرسوم عند إنشائه بمساحة سطح النموذج أو الأداة، فإذا تغيّرت هذه المساحة لا يشعر كائن الرسوم بهذا التغيير.. لهذا فإنّ

أنسب مكان تعرّف فيه هذا الكائن بالنسبة للنموذج هو حدث رسم النموذج Paint event، حتّى يعاد إنشاء كائن الرسوم كلّما حدث تغيير للنموذج يستدعي إعادة رسمه.

خصائص كائن الرسوم:

DpiX و DpiY:

تمثّل هاتان الخاصيتان عدد النقاط في كلّ بوصة Dots per inch (Dpi) أفقيّا (على المحور X) ورأسيّا (على المحور Y).

وحدات الصفحة PageUnit:

في الوضع التلقائيّ، تقاس كلّ الأطوال بالنقطة Pixel.. فإذا أردت تغيير ذلك، فاختر لهذه الخاصيّة أيّ وحدة قياس أخرى من المرقّم GraphicsUnit، الذي قيمه كالتالي:

Document	١,٣ بوصة.
Inch	بوصة.
Millimeter	ملليمتر.
Pixel	نقطة على شاشة الكمبيوتر (وهي القيمة الافتراضيّة).
Point	نقطة في ورقة الطباعة، وهي تساوي ١,٧٢ بوصة.
World	حدّد أنت القيمة التي تريد القياس بها.

كيفية رسم النصوص TextRenderingHint:

تأخذ هذه الوسيلة واحدة من قيم المرقّم TextRenderingHint، لإضافة تأثيرات معيّنة تحسّن من شكل النصّ المرسوم.

طراز التنعيم SmoothingMode:

مماثلة للخاصيّة السابقة، ولكنّها تنطبق على كلّ الأشكال، وليس النصوص فقط.. وهي تأخذ قيمة من قيم المرقّم SmoothingMode.

رسم الأشكال بوسائل كائن الرسوم:

يمتلك كائن الرسوم العديد من الوسائل التي تمكّنك من رسم الأشكال.. ولكي نشرح هذه الوسائل، تعالَ نعرّف كائن رسوم يرسم على النموذج بقلم أزرق:

```
Dim G As Graphics = Me.CreateGraphics
Dim P As New Pen(Color.Blue)
```

إنّ احتياجنا للقلم يرجع إلى أنّ أوّل معامل في وسائل رسم الأشكال هو القلم.. وطبعا باقي المعاملات هي إحداثيات الشكل.

دعنا نبدأ برسم مستطيل:

```
G.DrawRectangle(P, 10, 10, 200, 150)
```

والآن سنرسم قطري المستطيل:

```
G.DrawLine(P, 10, 10, 210, 160)
G.DrawLine(P, 210, 10, 10, 160)
```

جرّب ذلك في مشروع SimpleShapes.

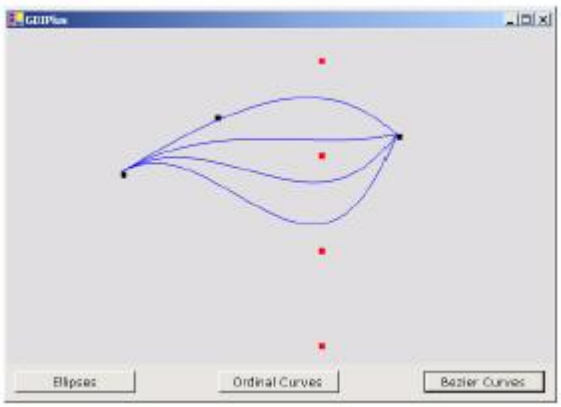
والجدول التالي يوضّح لك الوسائل التي ترسم الأشكال المختلفة (تلك التي تبدأ بالمقطع Draw)، والوسائل التي تملأ كل شكل منها بلون معيّن ــ إذا كان الشكل مغلقا بالطبع ــ (وهي تبدأ بالمقطع Fill)، مع ملاحظة أنّ استخدام وسيلة الملء يملأ الشكل إذا كان موجودا على النموذج، أو ترسمه ممتلئا باللون إذا لم يكن موجودا من قبل.

DrawArc:

ترسم قوسا.

DrawBezier:

ترسم منحنى بيزير، وهو منحنى له نقطتا بداية ونهاية، ويمرّ بنقطتين أخريين تتحكّمان في انحنائه، كما في الشكل:



DrawBeziers:

ترسم مجموعة من منحنيات بيزير معا.

DrawClosedCurve و **FillClosedCurve**:

ترسم منحنى مغلق.

DrawCurve:

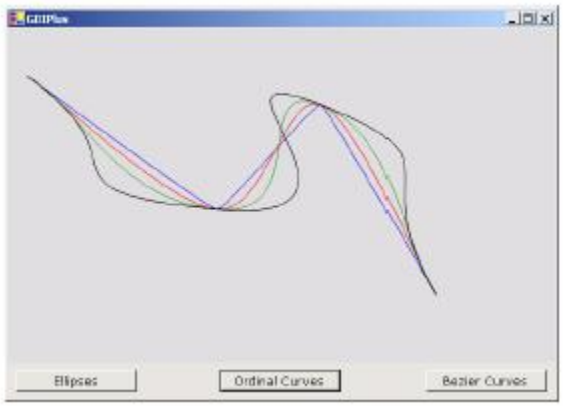
ترسم منحنى يمرّ بنقاط معيّنة (على الأقلّ ٤ نقاط، بداية ونهاية ونقطتــان على المنحنى):

Graphics.DrawCurve(القلم, _

درجة الانحناء, مصفوفة من النقاط التي يمرّ بها المنحنى)

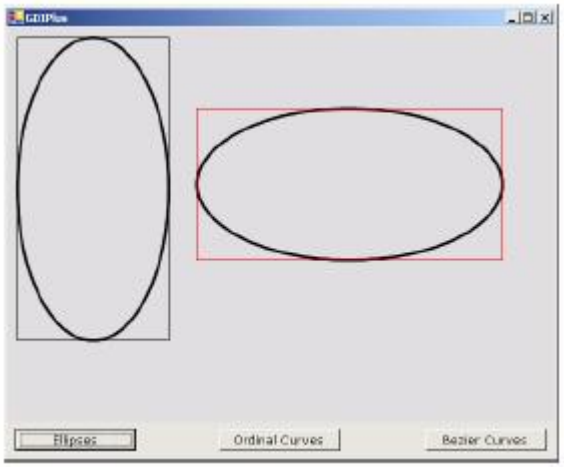
ولو لم ترسل المعامل الأخير، فسيتمّ رسم المنحنى بدرجة انحناء ١.

والشكل التالي يبيّن مجموعة من المنحنيات التي تمرّ بنفس النقاط، ولكــنّ لكلّ منها درجة انحناء مختلفة:



DrawEllipse و **FillEllipse:**

ترسم قطعا ناقصا، بمعلوميّة المستطيل الذي يحتويه.. والقطع الناقص هو شكل بيضاويّ له قطران غير متساويين (مساويان لضلعي المستطيل الذي يحتويه).



ونظرا لأنّه لا توجد وسيلة مستقلّة لرسم الدائرة، فيجب عليك أن تستخدم هذه الوسيلة لرسم الدوائر، وذلك برسم القطع الناقص داخل مربّع، وفي هذه الحالة سيكون طول ضلع المربّع مساويا لقطر الدائرة.. مثال: الجملة التالية ترسم دائرة زرقاء نصف قطرها ٥٠:

```
Graphics.DrawEllipse(Pens.Blue, _
    New Rectangle(50,20,100,100))
```

ويقع مركز هذه الدائرة في مركز المربّع الذي هو (٥٠ + [١٠٠] ÷ ٢ ، ٢٠ + [١٠٠] ÷ ٢)، أي النقطة (١٠٠ ، ٧٠).

وكقاعدة:

يقع مركز الدائرة عند النقطة (س + نصف طول ضلع المربّع ، ص + نصف طول ضلع المربّع).

DrawIcon:

تحمّل أيقونة داخل كائن الرسوم.. هذه الأيقونة سيتمّ مطّها لتملأ مساحة كائن الرسوم.

DrawIconUnstretched:

تحمّل أيقونة داخل كائن الرسوم، ولكن دون مطّها.

DrawImage:

تحمّل صورة داخل كائن الرسم، مع مطّها لتملأ مساحة كائن الرسوم.

DrawImageUnscaled:

تحمّل صورة داخل كائن الرسم، ولكن دون تغيير حجمها.

DrawLine:

ترسم خطّا يصل بين نقطتين.

DrawLines:

ترسم مجموعة من الخطوط.

DrawPath و FillPath:

ترسم جميع الأشكال الموجودة في كائن مسار الرسوم GraphicsPath object.. وسنتعرّف عليه لاحقا.

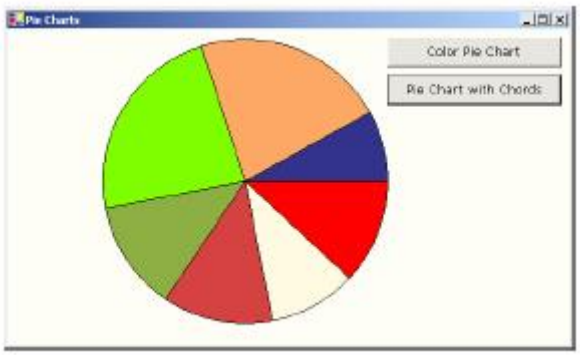
DrawPie و FillPie:

ترسم شريحة من الدائرة (قوس من قطع ناقص (أو دائرة) وضلعين يصلان نهايتي القوس بمركز الدائرة).. ولرسم هذا القوس أرسل لهذه

الوسيلة المستطيل الذي يحتوي على القطع الناقص، مـــع زاويـــة بدايـــة القوس، وزاوية انفراج ضلعي القوس.. مثال:

```
Graphics.DrawEllipse(Pens.Blue, _
    New Rectangle(50,20,100,300), 50, 30)
```

وأهمّ استخدام لهذا الشكل هو الرسوم البيانيّة التي توضّح النسب كشرائح ملوّنة من دائرة (مثل الرسم الذي يوضّح لك المساحة المشغولة والمساحة الفارغة من القرص الصلب في الويندوز).



DrawPolygon و FillPolygon:

ترسم مضلّعا مغلقا له أيّ عدد من الرءوس.. هذه الرءوس ترسلها لهـــذه الوسيلة في صورة مصفوفة من النقط.

DrawRectangle و FillRectangle:

ترسم مستطيلا.

DrawRectangles و FillRectangles:

ترسم مجموعة من المستطيلات.

DrawString:

ترسم نصّا بخطّ معيّن.

FillRegion:

لملء كائن من النوع "منطقة" Region بلون معيّن.

ولن تجد صعوبة في استخدام هذه الوسائل.. فقط لاحظ ما يلي:

- أوّل معامل في دوال رسم الأشكال هو كائن القلم Pen Object، وذلك لتحديد لون الخطّ وسمكه.
- أوّل معامل في دوال ملء وتلوين الأشكال المغلقة هو كائن الفرشاة Prush Object، وذلك لتحديد لون التلوين وطرازه.
- باقي معاملات وسائل الرسم والتلوين توضّح موضع الشكل ومقاييسه.
- الوسائل التي ترسم أكثر من شكل في نفس الوقت، تتطلّب ــ كمعامل ــ مصفوفة تعرّف مواضع ومقاييس هذه الأشكال.. فمثلا DrawRectangles تقبل مصفوفة تحتوي على كائنات من النوع "مستطيل" Rectangle objects.

كائن مسار الرسوم GraphicsPath Object:

يمثّل هذا الكائن رسما مغلقا يتكوّن من مجموعة من الخطوط والمستطيلات والمنحنيات.

وأبسط طرية لتكوين هذا الكائن، هي تعريف نسخة جديدة منه، واستخدام وسائله التالية لإضافة الرسومات إليه:

إضافة قوس **AddArc** ــ إضافة قطع ناقص **AddEllipse** ــ إضافة مضلّع **AddPolygon** ــ إضافة منحنى بيزير التكعيبيّ **AddBezier** ــ إضافة خطّ **AddLine** ــ إضافة مستطيل **AddRectangle** ــ إضافة منحنى **AddCurve** ــ إضافة منحنى مغلق **AddClosedCurve** ــ إضافة شريحة من دائرة **AddPie** ــ إضافة نصّ **AddString**.

وهناك المزيد من الوسائل تسمح لك بإضافة عدد من الأشكال دفعة واحدة، مثل:

إضافة منحنيات بيزير **AddBeziers** ــ إضافة خطوط **AddLines** ــ إضافة مستطيلات **AddRectangles**.

ليس هذا فحسب، بل يمكنك إضافة مسار رسومات آخر إلى المسار الحاليّ باستخدام الوسيلة **AddPath**.

طبعا لاحظت أنّ أسماء هذه الوسائل تتشابه مع أسماء وسائل الرسم بكائن الرسوم Graphics.. بل إنّهما يتشابهان في المعاملات.. والاختلاف الوحيد بينهما، هي أنّ وسائل الرسم في كائن الرسوم لها معامل زائد، هو المعامل الأوّل، الذي تحدّد فيه القلم الذي سترسم به.. ونظرا لأنّ كائن مسار الرسوم لا يرسم هذه الرسوم على النموذج أو الأدوات بنفسه، فإنّك لا تحتاج لتحديد القلم له.. ولرسم الأشكال الموجودة في هذا المسار

استخدم الوسيلة DrawPath الخاصّة بكائن الرسوم Graphics Object.. حينئذٍ يمكنك تحديد القلم الذي سترسم به.

وفي المثال التالي ننشئ مسار رسوم ونضيف له قطعا ناقصا ومنحنى بيزير:

```
Dim myPath As New GraphicsPath( )
myPath.AddEllipse(New Rectangle(10, 30, 40, 50))
mypath.AddPie(100, 50, 100, 200, 90, 10)
Me.CreateGraphics.DrawPath(Pens.Aqua, myPath)
```

إنّ هناك فوائدَ جمّةً لكائن مسار الرسوم، فقدرتك على رسم شكل معقّد يتكوّن من مجموعة مختلفة من الأشكال، والتعامل معه باعتباره شكلا واحدا، يمنحك القدرة على تكرار رسم هذا الشكل لأيّ عدد من المرّات باستدعاء وسيلة واحدة DrawPath، مع قدرتك على تغيير القلم (اللون والسمك) في كلّ مرّة، وقدرتك على ملء الشكل بألوان مختلفة أو صور مختلفة في كلّ مرّة.

وكذلك يستخدم مسار الرسوم في إنشاء كائن مسار متدرّج اللون PathGradient، وهو ما سيسمح لك برسم أشكال ساحرة، كما سنرى فيما بعد.

الألوان المتدرّجة Gradients

لدينا نوعان من التدرّج:

- المتدرّجات الخطّيّة Linear Gradients:
- المتدرّجات المتشعّبة Path Gradients:

المتدرّجات الخطّيّة Linear Gradients:

يتمّ إنشاء هذا النوع من الألوان المتدرّجة بالفرشاة LinearGradientBrush.. ابدأ بتعريف متغيّر من هذه الفرشاة كالتالي:

Dim lgBrush As New LinearGradientBrush(مستطيل,
طراز التدرّج ,لون النهاية ,لون البداية)

هذه الفرشاة مؤهّلة لملء مساحة مماثلة لمساحة المستطيل المحدّد في المعامل الأوّل، بألوان تتدرّج من لون البداية حتّى تصل للون النهاية.. فإذا كان الشكل الذي ستملؤه الفرشاة أصغر من المستطيل، فإنّ جزءا فقط من التدرّج هو الذي سيملأ الشكل.. أمّا لو كان الشكل أكبر من المستطيل، فسيتمّ تكرار التدرّج حتّى يملأ كلّ الشكل.

ويحدّد طراز التدرّج اتّجاه التدرّج، ويمكن أن يأخذ قيمةً من القيم التالية:

BackwardDiagonal	تدرّج قطريّ من أعلى اليمين إلى أسفل اليسار.
ForwardDiagonal	تدرّج قطريّ من أعلى اليسار إلى أسفل اليمين.
Horizontal	تدرّج أفقيّ.
Vertical	تدرّج رأسيّ.

والجزء التالي من الكود موجود في مشرع GDIPlusGradients فـــي مجلّد برامج هذا الفصل، وهو يملأ مستطيلين بألوان متدرّجة:

```
Dim G As Graphics = Me.CreateGraphics
Dim R As New RectangleF(20, 20, 300, 100)
' لون البداية أزرق بنفسجي
Dim startColor As Color = Color.BlueViolet
' لون النهاية أصفر فاتح
Dim EndColor As Color = Color.LightYellow
Dim LGBrush As New LinearGradientBrush (R,
    startColor, EndColor, _
    LinearGradientMode.Horizontal)
G.FillRectangle(LGBrush, New Rectangle(20, 20, 200,
    100))
G.FillRectangle(LGBrush, New Rectangle(20, 150,
    600, 100))
```

وتوجد عدّة صيغ أخرى لتعريف هذه الفرشاة.. فمثلا، يمكنـــك اســـتخدام نقطتين بدلا من المستطيل، هاتان النقطتان ســـتمثلان رأســـي المســـتطيل المتقابلين (العلويّ الأيسر، والسفليّ الأيمن).. وهما في نفس الوقت تحدّدان اتجاه التدرّج، حيث سيكون في اتجاه الخطّ الواصل بين النقطتـــين (ممّـــا سيتيح لك اتجاهات غير ممكنة في صيغة المستطيل وطراز التدرّج).

```
Dim lgBrush As New LinearGradientBrush(نقطة البداية,
    لون النهاية, لون البداية, نقطة النهاية)
```

كما أنّ هناك صيغة تتيح لك تحديد زاوية التدرّج، بدلا من طراز التدرّج:

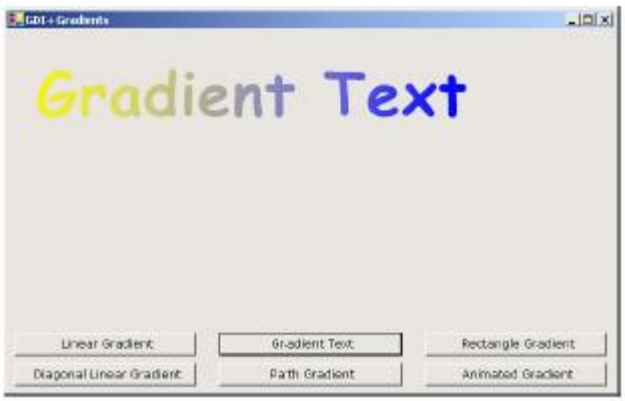
```
Dim lgBrush As New LinearGradientBrush(مستطيل,
    زاوية التدرّج, لون النهاية, لون البداية)
```

حيث تقاس هذه الزاوية بالنسبة للمحور الأفقيّ، في اتجاه عقارب الساعة.

ويمكنك إضافة معامل أخير على الصيغة السابقة.. لو جعلته True، فستتأثّر الزاوية بالتحويلات Transform المجراة على الإحداثيات (حيث يتمّ قياسها بالنسبة للمحاور بعد تدويرها)..أو False إذا كنت تريد أن يتمّ قياس الزاوية بطريقة مستقلّة.

ويمكن استخدام الألوان المتدرّجة لملء المنحنيات والمضلّعات الأخرى.. في هذه الحالة يتمّ ملء المستطيل الذي يحتوي المنحنى أو المضلّع، ولكن لا يتمّ عرض إلا الجزء الواقع داخل الشكل.

والطريف أنّك تستطيع رسم نصّ على النموذج بألوان متدرّجة.. هذا الكود أيضا موجود في مشروع GDIPlusGradients وهو يفعل ذلك:




```
Dim G As Graphics = Me.CreateGraphics
' مسح أيّ رسوم على النموذج، مع المحافظة على لون الخلفيّة
G.Clear (Me.BackColor)
G.TextRenderingHint = _
   System.Drawing.Text.TextRenderingHint.AntiAlias
Dim largeFont As New Font("Comic Sans MS", 48, _
   FontStyle.Bold, GraphicsUnit.Point)
Dim txt As String = "نصّ متدرّج"
```

```
' معرفة مساحة النصّ
Dim txtSize As SizeF = G.MeasureString(txt,
    largeFont)
' إنشاء مستطيل له مساحة النصّ
Dim R As New RectangleF(New PointF(0,0), txtSize)
' إنشاء الفرشاة
Dim grBrush As New LinearGradientBrush(R,
    Color.Yellow, Color.Blue,
    LinearGradientMode.BackwardDiagonal)
G.DrawString(txt, largeFont, grBrush, 20, 20)
```

الجدير بالذكر، أنّ المتدرّج الخطّي يمنحك هذه الإمكانيّات الإضافيّة:

١- استخدام أكثر من لونين للتدرّج:

وذلك باستخدام خاصيّة InterpolationColors.

٢- تحديد كيفيّة مزج الألوان:

وذلك باستخدام فئة المزج Blend Class أو الوسيلتين SetSigmaBellShape و SetBlendTriangularShape .

٣- تحويل إحداثيات المتدرّج:

وذلك باستخدام خاصيّة Transform.

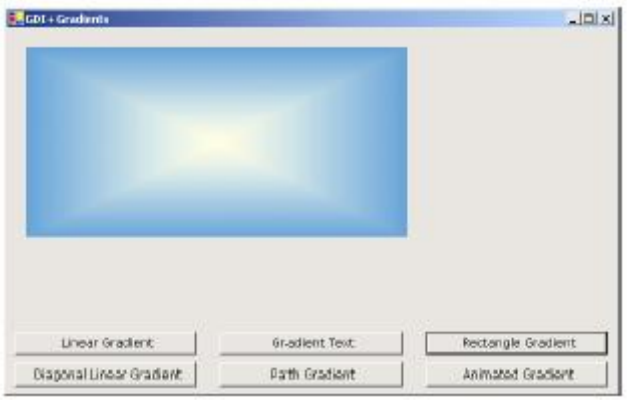
٤- إنشاء متدرّج يبدأ من لون في المركز، ويتدرّج إلى لون آخر على الطرفين.

٥- إنشاء متدرّج خطّي غير منتظم التدرّج Nonuniform Linear Gradient.

ويمكنك التعرّف على كلّ هذه الإمكانيّات، بالاستعانة بملفات المساعدة المرفقة باللغة.. ألا تعتقد أنّ الوقت قد حان لتعتمد على نفسك قليلا؟

المتدرّجات المتشعّبة Path Gradients:

يمكنك استخدام الفرشاة PathGradientBrush لرسم ألوان متدرّجة، تبدأ من لون معيّن، وتنتشر في اتجاهات مختلفة، لتنتهي في كل اتجـــاه منهـــا بلون مختلف.



ولاختبار هذه الإمكانية الرائعة، افحص المشروع GDIPlusGradients.

وهذه الفرشاة تناسب تلوين كائن مسار الرســـوم GraphicsPath، لهـــذا فإنّك تحدد لها الكائن الذي ستستخدم في تلوينه عند تعريفها:

```
Dim pgBrush As PathGradientBrush
pgBrush = New LinearGradientBrush(GraphicsPath )
```

والآن استخدم الخاصيتين التاليتين من خصائص هذه الفرشاة لتحديد ألوان التدرّج:

لون المركز CenterColor:

لتحديد لون المركز، الذي يبدأ منه التدرّج.. ولو لم تحدّد هذه الخاصـــيّة، فستكون قيمتها الافتراضيّة هي مركز الشكل الذي سيتمّ تلوينه.

الألوان المحيطة SurroundColors:

مصفوفة من الألوان بعدد رءوس الأشكال الموجودة في كائن مسار الرسوم.. مثال:

```
Dim Colors( ) As Color = {Color.Yellow, Color.Green,
    Color.Blue}
pgBrush.SurroundColors = Colors
```

وبعد أن تنشئ كائن الفرشاة وتضبط خصائصه، استدع الوسيلة FillPath الخاصّة بكائن الرسوم لتلوين الأشكال الموجودة في مسار الرسوم.

والمثال التالي يريك كيف تلوّن مستطيلا بألوان متدرّجة من الأحمر في مركزه، إلى الأصفر والأخضر والأزرق والـ Cyan في أطرافه.. ويمكنك تجربة هذا المثال في مشروع GDIPlusGradients.. لاحظ أنّنا رسمنا المستطيل كأربعة خطوط مغلقة وأضفناها لمسار الرسوم:

```
Dim G As Graphics  = Me.CreateGraphics
Dim path As New GraphicsPath( )
path.AddLine(New Point(10, 10), New Point(400, 10))
path.AddLine(New Point(400, 10), New Point(400,
    250))
path.AddLine(New Point(400, 250), New Point(10,
    250))
Dim pathBrush As New PathGradientBrush(Path )
pathBrush.CenterColor = Color.Red
Dim surroundColors( ) As Color = _
    {Color.Yellow, Color.Green, Color.Blue,
     Color.Cyan}
pathBrush.SurroundColors = surroundColors
G.FillPath(pathBrush, Path)
```

ولا يوجد ما يمنع أن تضيف مستطيلا (أو أيّ شكل آخر) لكائن مسار الرسوم ثم تلوّنه بالفرشاة متشعّبة التدرّج.

تحويل الإحداثيّات Coordinate Transformations:

لدينا ثلاثة أنواع من تحويلات الإحداثيّات:

١ - تغيير مقياس الرسم Scaling:

بحيث يتغيّر حجم الشكل تكبيرا أو تصغيرا.. وتقوم بذلك الوسيلة ScaleTransform من وسائل كائن الرسوم Graphics Object:

Graphics.ScaleTransformation(نسبة التكبير أفقيّا,
نسبة التكبير رأسيّا)

فإذا كانت النسبة أصغر من واحد كانت العمليّة تصغيرا، وإذا كانت أكبر من ١ كانت تكبيرا.

٢ - تغيير الموضع Translation:

بحيث نحرّك الشكل من موضعه.. وتقوم بذلك الوسيلة TranslateTransform من وسائل كائن الرسوم:

Graphics.TranslateTransform(مقدار الانتقال أفقيّا,
مقدار الانتقال رأسيّا)

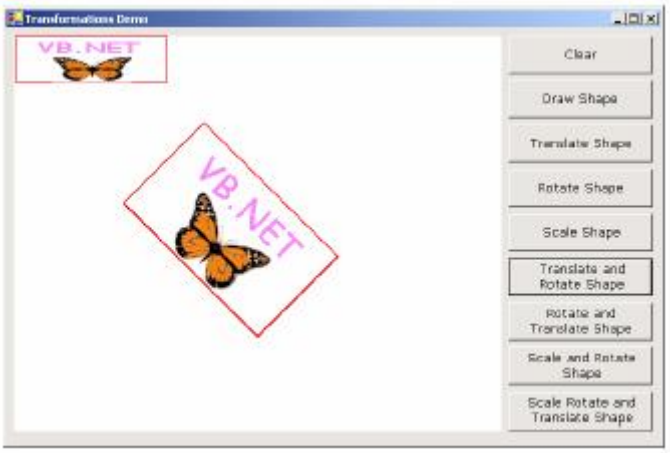
فإذا كان مقدار الانتقال موجبا انتقل الشكل لليمين أو لأسفل، وإذا كان سالبا، انتقل الشكل لليسار أو لأعلى.

١ - الدوران Rotation:

بحيث ندير الشكل بأيّ زاوية.. وتقوم بذلك الوسيلة RotateTransform من وسائل كائن الرسوم.

Graphics.RotateTransform(زاوية الدوران)

وتنحصر زاوية الدوران بين ٠ و ٣٦٠ درجة.



إنّ هذه التحويلات تراكميّة، فمثلا: تدوير الشكل ٣٠ درجة، ثمّ تدويره ٤٠ درجة، يكافئ تدويره ٧٠ درجة مباشرة.

هذا التراكم ناتج من أنّ هذه التحويلات يتمّ حفظها في مصفوفة تسمّى "مصفوفة التحويلات" Transformation matrix.

فإذا أردت أن تعيد الإحداثيّات لوضعها الأصليّ، فاستخدم الوسيلة ResetTransform.

ويمكنك اختبار هذه التحويلات في المشروع GDIPlusTransformations.

قص منطقة الرسم Clipping:

إذا أردت أن تقصّ منطقة من النموذج أو الأداة لتكون هي المنطقة التي يرسم عليها كائن الرسوم، فاستخدم الوسيلة SetClip الخاصّة بكائن الرسوم Graphics object.

وأوّل معامل تأخذه هذه الوسيلة، يحدّد منطقة الرسم.. في هذا الحالة يمكنك أن تجعل المعامل مستطيلا.. فإذا شئت أن تجعل للمنطقة أيّ شكل أكثر تعقيدا، فاستخدم كائن مسار الرسوم GraphicsPath، أو منطقة Region (المنطقة هي سجلّ يتكوّن من أشكال بسيطة، حيث تتكوّن المنطقة من اتحاد هذه الأشكال، أو من تقاطعها، أو من فرق أحدها من الآخر، أو من المنطقة التي تقع خارج التقاطع).

وهناك معامل آخر اختياريّ، يحدّد كيف ستتم إضافة هذه المنطقة للنموذج أو المنطقة المعرّفة سابقا في النموذج.. ويأخذ هذا المعامل قيمة من قيم المرقّم CombineMode، وهي:

Complement	منطقة الرسم الجديدة هي التي تقع خارج اتحاد المنطقتين.
Exclude	منطقة الرسم الجديدة هي التي تقع في المنطقة الجديدة وحدها، خارج منطقة التقاطع.

Intersect	منطقة الرسم الجديدة هي منطقة التقاطع.
Replace	المنطقة الجديدة ستكون بديلة لسابقتها.
Union	منطقة الرسم الجديدة هي ناتج اتّحاد المنطقتين.
XOR	منطقة الرسم الجديدة هي ناتج اتحاد المنطقتين ما عدا منطقة تقاطعهما.

وبعد تحديد منطقة الرسم، فإنّ أيّ شكل ترسمه لن يُرسم إلا في هذه المنطقة، وأيّ تحويل للإحداثيّات لن ينطبق إلا على هذه المنطقة.

ولإزالة المنطقة المقصوصة، والعودة للرسم على النموذج كلّه، استخدم الوسيلة ResetClip.

ويريك مشروع Clipping، كيف تعرض النصّ والصورة داخل قطع ناقص.

رسم النصوص:

لرسم نصّ على نموذج أو أداة، استخدم الصيغة التالية:

Graphics.DrawString(النصّ, الخطّ, الفرشاة, X, Y)

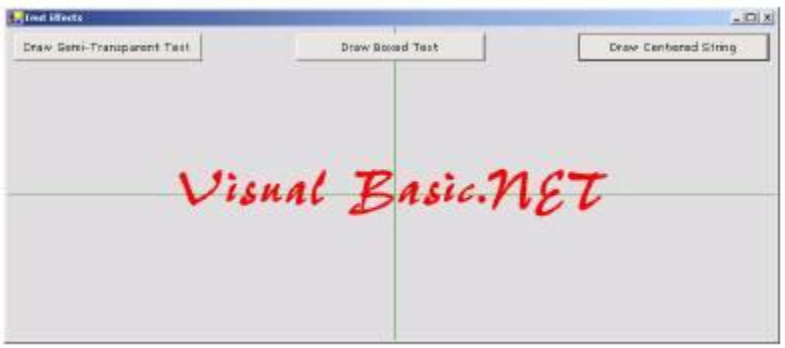
حيث X و Y يمثّلان نقطة بداية الكتابة (أعلى نقطة يسارا في المنطقة التي تحتوي النصّ).

ولمعرفة المساحة التي يشغلها النصّ على النموذج أو الأداة، استخدم الوسيلة "قياس النصّ" MeasureString، حيث تعيد لك كائن حجم بدقّة عشريّة SizeF، بحيث يمكنك استخدام خاصيتيه العرض والارتفاع لمعرفة مساحة النصّ.

والمثال التالي يوضّح لك كيف النصّ في مركز النموذج تماما:

```
Dim textSize As SizeF
Dim X As Integer, Y As Integer = 0
textSize = Me.Graphics.MeasureString(string, font)
X = (Me.Width – textSize.Width) / 2
G.DrawString("هذا النصّ في مركز النموذج", font, brush,
    X, Y)
```

ويمكنك تجربة ذلك في مشروع TextEffects في برامج هذا الفصل.



وهناك صيغة أخرى ترسم النصّ داخل المستطيل الذي تحدّده لها، حتّى لو اقتضى الأمر كتابة النصّ على أكثر من سطر داخل المستطيل (إذا كان النصّ أطول من عرض المستطيل):

Graphics.DrawString(النصّ, الخطّ, الفرشاة, مستطيل)

Graphics.DrawString(النصّ, الخطّ, الفرشاة, مستطيل, تنسيق النصّ)

ولمعرفة عدد السطور التي كتب فيها النصّ، استخدم الوسيلة MeasureString.. طبعا ستتعجب: ألم نستخدم هذه الوسيلة لمعرفة أبعاد النصّ؟

نعم، وهناك صيغة أخرى منها تنتهي بمعاملين مرجعيّين، يرجعان لك عدد سطور النصّ، وعدد الأعمدة التي كتب عليها:

e.Graphics.MeasureString(النصّ, الخطّ,
كائن حجم به أبعاد المستطيل الذي يحتوي النصّ ,_
تنسيق النصّ, عدد السطور, عدد الأعمدة)

ويمكنك تجربة ذلك في مشروع TextEffects في مجلّد برامج هذا الفصل.

ولكن ما هو المعامل تنسيق النصّ، ذلك الذي ظهر في أكثر من وسيلة؟ هذا المعامل من النوع StringFormat، وهو يمنحك الخصائص التالية:

محاذاة النصّ Alignment:

تأخذ هذه الخاصيّة واحدة من قيم المرقّم StringAlignment، وهي:

Center	يتم توسيط النصّ في مركز المستطيل.
Far	تتمّ محاذاة النصّ بعيدا عن ركن المستطيل العلويّ الأيسر.
Near	تتمّ محاذاة النصّ قريبا من ركن المستطيل العلويّ الأيسر.

قصّ النصّ Trimming:

تستخدم هذه الخاصيّة لتوضيح كيف سيتمّ قصّ النصّ إذا تجاوز طوله حدود المستطيل.. وتأخذ هذه الخاصيّة القيم التالية:

Character	يتمّ قصّ النصّ عند أقرب حرف لحدود المستطيل.
EllipsisCharacter	يتمّ قصّ النصّ عند أقرب حرف لحدود المستطيل، وتوضع بعض النقاط في نهاية النصّ للدلالة على أنّ باقي النصّ غير ظاهر.
EllipsisPath	يتمّ حذف جزء من منتصف النصّ، وتوضع مكانه بعض النقاط.
Word	يتمّ قصّ النصّ عند أقرب كلمة لحدود المستطيل.
EllipsisWord	يتمّ قصّ النصّ عند أقرب كلمة لحدود المستطيل، وتوضع بعض النقاط في نهاية النصّ للدلالة على أنّ باقي النصّ غير ظاهر.
None	لا يتمّ قصّ النصّ.

مؤشّرات التنسيق FormatFlags:

يمكن لهذه الخاصيّة أن تأخذ قيمةً أو أكثر من قيم المرقّم StringFormatFlags، ومن أهمّها DirectionRightToLeft لتحديد أنّ اتجاه النصّ من اليمين لليسار، DirectionVertical لكتابة النصّ رأسيّا.

والمثال التالي يريك كيف تكتب رأسيّا على النموذج:

```
Dim G As Graphics = Me.CreateGraphics
Dim SF As New StringFormat( )
SF.FormatFlags =
    StringFormatFlags.DirectionVertical
G.DrawString("Visual Basic", Me.Font, Brushes.Red,
    80, 80, SF)
```

التعامل مع الصور

مربّع الصورة PictureBox:

يمنحك VB أداةً متخصّصةً في عرض الصور والتعامل معها، تلك هي مربّع الصورة PictureBox.. ولكي تعرض صورةً فيه، استخدم خاصيّة "صورة" Image في نافذة الخصائص، حيث ستجد زر انتقال Ellipsis Button في خانة القيمة.. اضغط هذا الزر ليظهر لك مربّع حوار فتح ملفّ.. اختر الصورة التي تريد عرضها واضغط موافق.. هذه الصورة سيتمّ حفظها في الملفّ الذي يحمل نفس اسم النموذج ولكن امتداده "resx".. لهذا لن تحتاج لتوزيع الصورة الأصليّة مع برنامجك، فهي محتواةٌ فيه بالفعل.

ومن خصائص مربّع الصورة الهامّة ما يلي:

طراز الحجم SizeMode:

تمكّنك هذه الخاصيّة من اختيار كيفيّة ظهور الصورة ومحاذاتها في مربّع الصورة.. ولهذه الخاصيّة القيم التالية:

Normal	تظهر الصورة طبيعية كما هي بدون تعديل.. فإذا كانت أكبر من مربّع الصورة، فسيختفي جزء منها، وإذا كانت أصغر، فلن تملأ كل مساحة مربّع الصورة.
CenterImage	سيتمّ عرض الصورة بحيث تتوسّط مربّع الصورة.
StretchImage	سيتمّ مطّ الصورة أو تقليصها بحيث تشغل مساحة مربّع الصورة بالضبط.. ورغم أنّ هذه الطريقة هي أفضل طريقة لعرض الصورة، إلا إنّ عليك أن تراعي تناسب أبعاد مربّع الصورة مع أبعاد الصورة، حتّى لا تتشوّه الصورة عند مطّها.. افحص مشروع ImageLoad في مجلّد برامج هذا الفصل، لترى كيف نحافظ على تناسب مقاييس الصورة.
AutoSize	سيتمّ تغيير أبعاد مربّع الصورة ليطابق أبعاد

الصورة.. إنّ هذا الاختيار غير مفضّل، لأنّ وضع صورة كبيرة في مربّع الصورة سيجعله يغطّى باقي الأدوات الموجودة على النموذج.

طراز الحافّة BorderStyle:

وهي تأخذ قيم المرقّم BorderStyle وهي: ثابتة مجسّمة **Fixed3D** ــ ثابتة مفردة **FixedSingle** ــ بدون حافّة **None**.

الصورة Image:

تسمح لك بتحديد الصورة التي تعرضها في مربّع الصورة.

صورة الخلفيّة BackgroundImage:

تسمح لك بتحديد الصورة التي تعرضها في خلفيّة مربّع الصورة.

كائن الصورة Image Object

مرارا نكرّر: إنّ كلّ شيء في VB.Net ما هو إلا كائن، لهذا فمن الطبيعيّ أن يكون هناك كائن يمكنك من خلاله التعامل مع الصورة.
فمثلا، خاصيّة Image الخاصّة بمربّع الصورة تمثّل كائن صورة Image object.. هذا الكائن يمثّل الصورة الموجودة في مربّع الصورة، ويمنحك الخصائص والوسائل اللازمة للتعامل معه.
ويمكن أن تعرّف كائن صورة، وتضع فيه الصورة التي تحملها خاصيّة Image كالتالي:

```
Dim img As Image = PictureBox1.Image
```

كما يمكنك أن تحمّل صورة من ملفّ وتضعها في كائن صورة عن طريق الوسيلة "من ملفّ" FromFile ثمّ تضعها في مربّع الصورة كالتالي:

```
Dim img As Image = Image.FromFile("Butterfly.jpg")
PictureBox1.Image = img
```

خصائص كائن الصورة:

الخصائص التالية للقراءة فقط:

الدقّة الأفقيّة HorizontalResolution

والدقّة الرأسيّة VerticalResolution:

يمكنك بهاتين الخاصيّتين أن تعرِف دقّة تمثيل الصور (عدد النقط في كل بوصة Pixels -per-inch).

العرض Width والارتفاع Height:

أعتقد أنّ هاتين الخاصيتيّن واضحتان.. ولكنّ ما سألفت نظرك إليه هنـــا، هو التالي:

Width ÷ HorizontalResolution = عرض الصورة بالبوصة.

Height ÷ VerticalResolution = ارتفاع الصورة بالبوصة.

كما أحبّ أن ألفت نظرك أيضا إلى أنّ تغيير عـــرض وارتفـــاع مربّـــع الصورة، لا يؤثّر على عرض وارتفاع كائن الصورة، فهاتان الخاصيتان تشيران دائما إلى مقاييس الصورة الأصليّة، وليس النسخة المعروضة في مربّع الصورة.

تنسيق النقط PixelFormat:

هناك العديد من تنسيقات النقط، يجمعها المرقّم PixelFormat.. فهنـــاك مثلا القيمة Format24bppRgb، وهي تدلّ على أنّ كـــل نقطـــة فـــي الصورة تمثّل اللون بـــ ٢٤ خانة ثنائيّة 24 bits per pixel.

الوسائل:

تدوير وعكس RotateFlip:

تمكّنك هذه الخاصيّة من تدوير الصورة أو عكسها أو كلا الأمرين معا.

Image.RotateFlip(النوع)

حيث النوع هو إحدى قيم المرقّم RotateFlipType التالية:

RotateNoneFlipNone	لا تدوير ولا عكس.. واضح طبعا أنّك بهذا لن تفعل شيئا في الصورة (يفيد هذا في إلغاء عمليات التدوير والعكس السابقة، لإعادة الصورة إلى وضعها الأصليّ).
Rotate90FlipNone	تدوير الصورة بزاوية ٩٠ بدون عكسها.
Rotate90FlipX	تدوير الصورة بزاوية ٩٠ وعكسها أفقيّا.
Rotate90FlipXY	تدوير الصورة بزاوية ٩٠ وعكسها أفقيّا ورأسيّا.
Rotate90FlipY	تدوير الصورة بزاوية ٩٠ وعكسها رأسيّا.

وأعتقد أنّك تستطيع فهم باقي هذه التعبيرات:

Rotate180FlipNone ـ Rotate180FlipX ـ

Rotate180FlipXY ـ Rotate180FlipY ـ

Rotate270FlipNone ـ Rotate270FlipX ـ

Rotate270FlipXY ـ Rotate270FlipY ـ

RotateNoneFlipX ـ RotateNoneFlipXY ـ

RotateNoneFlipY.

مثال:

```
PictureBox1.Image.RotateFlip(
        RotateFlipType.RotateNoneFlipY)
PictureBox1.Refresh( )
```

لاحظ استخدام خاصيّة الإنعاش لإعادة رسم الصـــورة، وإلا فلـــن يـــرى المستخدم أيّ تغيير.

ويمكنك التدريب على هذه الوسيلة في مشروع LoadImage.

كوّن صورة مصغّرة GetThumbnailImage:

تمكّنك هذه الوسيلة من الحصول على نسخة مصغّرة من الصورة، حيـــث يمكنك استخدامها في عرض "كتالوج" الصور للمستخدم، لتسمح له باختيار صورة منها لعرضها كاملة.. ولهذه الوسيلة الصيغة التالية:

Image.GetThumbnailImage(العرض, الارتفاع, إلغاء, بيانات)

حيث العرض والارتفاع يحدّدان بُعدَيِ الصورة المصغّرة الناتجة، بينمـــا إلغاء وبيانات يستخدمان في حالة إلغاء العمليّة، ونظرا لأنّ تكوين الصورة المصغّرة لا يستغرق وقتا، فسنجعل هذين المعاملين دائما Nothimng..

مثال:

```
PictureBox1.Image = img.GetThumbnailImage(32, 32,
    Nothing, Nothing)
```

ولديك في مجلّد برامج هذا الفصل، المشروع Thumbnails، وفيه تسمح للمستخدم باختيار مجلّد، حيث يقوم البرنامج بفحص ملفّات هـــذا المجلّـــد وعرض صور مصغّرة على النموذج لكلّ الصور الموجودة بهذا المجلّد.. ويتمّ عرض كلّ صورة مصغّرة في مربّع صورة خاصّ بها، حيـــث يـــتمّ إنشاؤه في وقت التنفيذ وضبط إحداثياته.. وعندما يضـــغط المســـتخدم أيّ صورة مصغّرة، يتمّ عرض صورتها الأصليّة كاملة في نموذج آخر.

هذا هو الكود الذي يعرض الصور المصغّرة:

```
Dim file As String
Dim FI As FileInfo
Dim PBox As PictureBox, img As Image
Dim Left As Integer = 280
Dim Top As Integer = 40
Dim ctrl As Integer
' احذف كلّ مربّعات الصور الموجودة على النموذج
For ctrl = Me.Controls.Count - 1 To 2 Step -1
    Me.Controls.Remove(Me.Controls(ctrl))
Next
' استخدم الوسيلة التالية لمحو أيّ رسومات غير دائمة من على النموذج
Me.Invalidate( )
For Each file In Directory.GetFiles(
        Directory.GetCurrentDirectory)
    FI = New FileInfo(file)
    If FI.Extension = ".GIF"
       Or FI.Extension = ".JPG" Or _
       FI.Extension = ".BMP" Then
        ' إنشاء مربّع صورة جديد
        PBox = New PictureBox( )
```

```
' تحميل الصورة
img = Image.FromFile(FI.FullName)
' وضع الصورة المصغّرة في مربّع الصورة
PBox.Image = img.GetThumbnailImage(64,
    64, Nothing, Nothing)
' إذا تجاوز موضع اليسار لمربّع الصورة الجديد عرض النموذج،
' فضعه في بداية صفّ تالي (بزيادة موضع حافته العليا)
If Left > 580 Then
        Left = 280
        Top = Top + 74
End If
' ضبط إحداثيات مربّع الصورة
PBox.Left = Left
PBox.Top = Top
PBox.Width = 64
PBox.Height = 64
PBox.Visible = True
' لا بدّ من الاحتفاظ باسم ملفّ الصورة،
' حتّى يمكن عرضها عند الضغط على الصورة المصغّرة
PBox.Tag = FI.FullName
' عرض مربّع الصورة على النموذج
Me.Controls.Add(PBox)
' إنشاء مستجيب لحدث ضغط مربّع الصورة
AddHandler PBox.Click, New _
        System.EventHandler(AddressOf
        OpenImage)
```

```
            إضافة عرض مربّع النصّ + ١٠ (مسافة فاصلة) '
            إلى المتغيّر الدال على موضع مربّع الصورة الجديد '
            Left += 74
        End If
Next
```

وهذا هو الكود الذي يعرض الصورة عند الضغط على صورتها المصغّرة:

```
Dim imgForm As New previewForm( )
imgForm.PictureBox1.Image = Image.FromFile(
    sender.tag)
imgForm.Show( )
```

حفظ Save:

استخدم هذه الوسيلة لحفظ الصورة الموجودة في كائن الصورة، كملفّ على الجهاز.. مثال:

```
PictureBox1.Image.Save("c:\tmpImage.bmp")
```

ويمكنك اختيار النوع الذي تريد حفظ الصورة به، كالتالي:

```
PictureBox1.Image.Save("c:\tmpImage.bmp", نـــوع
الصورة)
```

حيث نوع الصورة هو إحدى قيم المرقّم System.Drawing.Imaging.ImageFormat.. واضح أنّ الاسم طويل، وطبعا تعرف أنّك تستطيع كتابة جملة الاستيراد التالية في مشروعك:

```
Imports System.Drawing.Imaging
```

وبذلك تستطيع استخدام اسم المرقّم مختصرا.

ولهذا المرقّم القيم التالية:

Bmp	Bitmap، وهو أبسط أنواع الصور، حيث يتمّ حفظ الصورة في هيئة مصفوفة ثنائيّة البعد (جدول من صفوف وأعمدة).. وبهذا فإنّ كل خانة تحمل الإحداثيّات المناظرة لنقطة في الصورة (رقمها في الصف يناظر الإحداثيّ الأفقيّ، ورقمها في العمود يمثّل الإحداثيّ الرأسّي)، بينما قيمة الخانة تمثّل لون النقطة.
Emf	Enhanced Windows metafile.
Exif	Exchangeable Image Format.
Gif	Graphics Interchange Format.
Icon	أيقونة.
Jpeg	Joint Photographic Experts Group.
MemoryBmp	لحفظ الصورة كمصفوفة ذاكرة.
Png	W3C Portable Network Graphics.
Tiff	Tagged Image File Format.
Wmf	Windows metafile.

واضح طبعا مدى ثراء أنواع الصور التي يتعامل معها VB.Net، أكّثر ممّا في VB6.

الصور ولوحة القصاصات Clipboard:

رأينا من قبل كيف ننسخ النصوص إلى لوحة القصاصات، حتّى نستطيع أن نلصقها في أيّ تطبيق ويندوز آخر أو في أيّ موضع من هذا التطبيق.
كلّ ما فعلناه حينئذ هو استخدام الوسيلتين "اقرأ كائن البيانات" GetDataObject و"احفظ كائن البيانات" SetDataObject.
لا جديد هنا.. سنستخدم نفس الوسيلتين، ولكن مع الصور هذه المرة.
استخدم الجملة التالية لنسخ الصورة المعروضة في مربّع الصور إلى لوحة القصاصات:

```
Clipboard.SetDataObject(PictureBox1.Image)
```

ولو أحببت أن تبقى الصورة في لوحة القصاصات بعد إغلاق برنامجك فاستخدم الصيغة:

```
Clipboard.SetDataObject(PictureBox1.Image, True)
```

ولاستعادة الصورة من لوحة القصاصات، يمكنك استخدام الوسيلة GetDataObject.. هذه الوسيلة تُرجع كائنا من النوع "كائن واجهة البيانات" IDataObject، وهو يمنحك الوسيلتين:

اقرإ البيانات GetData:

تعيد لك محتويات لوحة القصاصات.. ونظرا لأنّ لوحة القصاصات قد تحتوي على بيانات من أنواع مختلفة معا في نفس الوقت، فإنّك تمدّ هذه الوسيلة بمعاملٍ يمثّل نوع البيانات الذي تريده.

هل البيانات موجودة GetDataPresent:

ترجع True إذا كان نوع البيانات الذي أرسلته لها كمعامل هو نوع البيانات الموجودة في لوحة القصاصات.. ويمكنك أن ترسل لهذه الوسيلة معاملا ثانيا، إذا جعلت قيمته True، فإنّ لوحة القصاصات تحاول تحويل البيانات الموجودة بها إلى النوع الذي أرسلته في المعامل الأوّل.

ويمكنك استخدام الجملة التالية لعرض الصورة من لوحة القصاصات في مربّع الصورة:

```
PictureBox1.Image = _
    Clipboard.GetDataObject.GetData(DataFormats
    .Bitmap)
```

ولمزيد من التدريب، لديك مشروع ImageClipboard في مجلّد برامج هذا الفصل.

رسم صورة DrawImage:

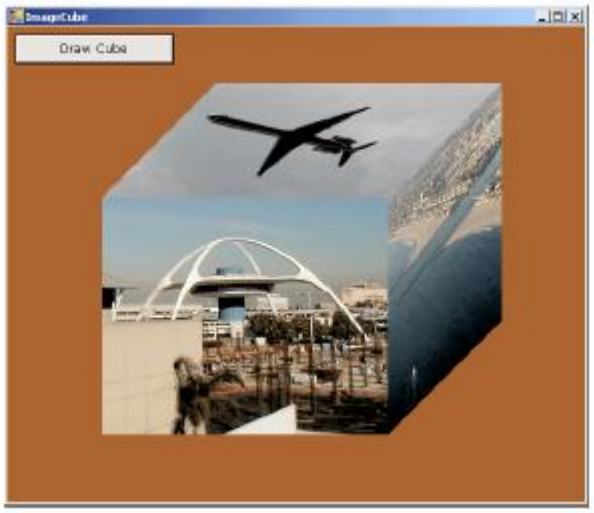
يمنحك كائن الرسوم Graphics الوسيلة DrawImage لتمكّنك من رسم الصور على أيّ سطح تريده.. ولهذه الوسيلة ٣٠ صيغة مختلفة، قد تحتاج لكتاب بمفردها!!.. لهذا سنكتفي بشرح أربع صيغ:

Graphics.DrawImage(الصورة, نقطة رأسها العلويّ الأيسر)

Graphics.DrawImage(الصورة, المستطيل الذي يحتويها)

Graphics.DrawImage(الصورة, points( ))

وفي الصيغة الثالثة، المعامل الثاني هو مصفوفة نقط، حيث تضع فيها ثلاثة نقاط، تمثّل ثلاثة من رءوس متوازي الأضلاع، الذي تريد أن تحوّر الصورة لترسم بداخله.. هذه الرءوس الثلاثة هي العلويّ الأيسر والعلوي الأيمن، والسفليّ الأيسر.. أمّا الرأس الرابع فيمكن استنتاجه، ولا حاجة بك لإرساله.. ولديك في مجلّد برامج هذا الفصل، المشروع المثير ImageCube، وفيه تعرف كيف يمكنك رسم مكعّب ثلاثيّ الأبعاد، ووضع صورة على كلّ وجه من أوجهه الثلاثة المواجهة للمستخدم.. إنّ فعل هذا بسيط جدّا، فأنت تعرف أنّك ترسم المكعّب بثلاثة متوازيات أضلاع، أحدها في المواجهة، والثاني يعطي الإحساس بالوجه الجانبيّ، والثالث يعطي الإحساس بقمّة المكعّب.. اختبر هذا المشروع الشيّق.

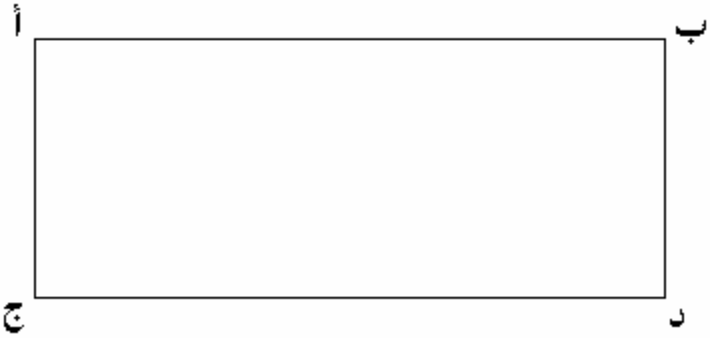


وهناك ملاحظة طريفة أخرى بخصوص هذه الصيغة.. إنّ بإمكانك أن (تلخبط) ترتيب مصفوفة النقط، وبذلك تقلب الصورة أو تعكسها، نتيجة تغيير مواضع رءوسها!.. طبعا لا يشترط دائما أن تعرض الصورة في متوازي أضلاع.. تذكّر فقط أنّ المستطيل هو متوازي أضلاع قائم الزاوية!

ويمكنك أن تجعل رأسين متطابقين لترسم الصورة في مثلث!

أبدع كما يحلو لك، فعلى حسب ترتيب رءوس المضلّع، سيتمّ قلب الصورة أو تدويرها أو عكسها أو مطّها أو تقليصها.. المهمّ أن تعرف ما تفعله!!

افترض أنّ مستطيل الصورة هو أ ب ج د، حيث أ هو الرأس العلويّ الأيسر، و ب هو الرأس العلويّ الأيمن، و ج هو الرأس السفليّ الأيسر، و د هو الرأس السفليّ الأيمن.



الجدول التالي يوضّح لك بعض طرق التلاعب بهذه الصورة، على حسب ترتيب إضافتك للنقاط للمصفوفة:

ترتيب مصفوفة النقط	التأثير
أ ب ج	الصورة كما هي.
ج د أ	عكس الصورة بالنسبة للمحور الأفقيّ.
ب أ د	عكس الصورة بالنسبة للمحور الرأسيّ.
د ب ج	عكس الصورة بالنسبة للقطر ب ج.
أ ج د	عكس الصورة بالنسبة للقطر أ د.
ج أ د	تدوير الصورة ٩٠ درجة في اتجاه عقارب الساعة.
ب د أ	تدوير الصورة ٩٠ درجة عكس اتجاه عقارب الساعة.
أ ج ج	رسم الصورة في مثلث!

كما أنّك تستطيع استخدام أيّ نقط أخرى تقع داخل الصورة أو خارجها، لرسم شرائح ممطوطة أو مقلّصة من الصورة!
وهكذا.. يمكنك أن تفعل بالصورة ما تشاء.

أمّا الصيغة الرابعة فهي تسمح لك بتحديد سمات الصورة:

Graphics.DrawImage(الصورة, points( ), مستطيل,
(السمات ,وحدة القياس

ويمثّل المستطيل الجزء الذي تريد رسمه من الصورة.. وتمثّل وحدة القياس الوحدة التي ستقاس بها أبعاد المستطيل.
أمّا المعامل الأخير فهو كائن من النوع ImageAttributes، حيث يمنحك بعض الوسائل لتغيير سمات الصورة، مثل جعلها تظهر على النموذج بالتدريج، ومثل تصحيح ألوانها.
ابدأ بتعريف كائن من هذا النوع، مع ملاحظة أنّه ينتمي لفضاء الاسم System.Drawing.Imaging، فاستورده لو أردت.

Dim attr As
New System.Drawing.Imaging.ImageAttributes

ومن وسائل هذا الكائن:

تغيير إضاءة الصورة SetGamma:

إذا تركت هذه الخاصيّة بالقيمة ١، فلن يتغيّر شيءٌ في لون الصورة، ولكن لو جعلتها أكبر من ١ فستصير ألوان الصورة فاتحةً أكثر، وإذا جعلتها أصغر من ١فستصير ألوان الصورة أغمق.

والكود التالي يريك كيف ترسم صورة في مربّع الصورة، مع تفتيح لونها بنسبة ٢٥%:

```
Dim attrs As
    New System.Drawing.Imaging.ImageAttributes( )
attrs.SetGamma(1.25)
Dim dest As New Rectangle(0, 0, _
        PictureBox1.Width, PictureBox1.Height)
G.DrawImage(img, dest, 0, 0, img.Width, img.Height,
        GraphicsUnit.Pixel, attrs)
```

هذا بافتراض أنّ G هو كائن رسوم يرسم في مربّع الصورة، و img هو كائن صورة تمّ تحميل الصورة به.

قنوات فصل اللون SetOutputChannel:

تقنية فصل الألوان هي تقنية معروفة لزيادة جودة طباعة الصور الملوّنة، حيث تتمّ طباعة الصورة على نفس الورقة أربع مرّات.. في كلّ مرّة يتمّ طباعة لون معيّن تمّ فصله من الصورة بمفرده.. وفي الغالب تُفصل الصورة إلى الألوان: cyan و البنفسجيّ المحمرّ magenta والأصفر yellow والأسود black.

وللحصول على هذه النسخ من الصورة، استدع هذه الوسيلة أربع مرّات بالمعاملات التالية:

```
attrs.SetOutputChannel(
        ColorChannelFlag.ColorChannelC) ' cyan
attrs.SetOutputChannel
(ColorChannelFlag.ColorChannelM) ' magenta
attrs.SetOutputChannel(
        ColorChannelFlag.ColorChannelY) ' yellow
```

```
attrs.SetOutputChannel(
    ColorChannelFlag.ColorChannelK) ' black
```

وهناك صيغة أخرى كالتالي:

```
attrs.SetOutputChannel(
    ColorChannelFlag.ColorChannelLast)
```

وذلك لاستدعاء الوسيلة مرّة أخرى بآخر لون تمّ استدعاؤها به.

وبخصوص باقي صيغ وسيلة رسم الصورة DrawImage، فأرجو أن تتعرّف عليها، إمّا عن طريق ملفات المساعدة، أو عن طريق تلميح الشاشة الذي يظهر عند كتابة اسم الوسيلة في محرّر الكود.. مع ملاحظة أنّ معظم هذه الوسائل تتلقّي معاملا، عبارة عن مستطيل يمثّل المساحة التي سترسم فيها الصورة.. فإذا كانت أبعاد الصورة مختلفة عن أبعاد المستطيل، يتمّ مطّها أو تقليصها لتتناسب معه.

كما أنّ بعض الصيغ تتلقّي مستطيلين: الأوّل يحدّد الجزء الذي تريد رسمه من الصورة، والآخر يوضّح المساحة التي سترسم هذا الجزء فيها.

الصور النقطيّة Bitmaps

تركيب الصورة:

الصورة هي مصفوفة من الألوان مرتّبة في صفوف وأعمدة.. ونتيجة للألوان وترتيبها، تبدو للعين كأنّها صورة متصلة مفعمة بالتفاصيل.

وهنا يجب توضيح الفارق بين كائن الصورة Image، وكائن الصورة النقطيّة Bitmap.. فكائن الصورة Image ثابت Static، بمعنى أنّك لا تستطيع تغيير نقاطه كلّ على حدة.. أمّا كائن الصورة النقطيّة Bitmap، فهو يسمح لك بالتعامل مع الصورة نقطة نقطة.. وهذه هي الطريقة التي يمكنك بها تعديل جزء من الصورة.. هذه باختصار هي عمليّة التعامل مع الصور Image Processing.. وإن كان من واجبي هنا أن أحذّرك، أنّ التعامل مع الصورة نقطة نقطة مضيعة للوقت، خاصّة أنّ العمليات التي تريد إجراءها على الصورة غالبا ما تكون معقّدة، مثل التعرّف على الأشكال والملامح أو البصمات.. فمثلا في هذه الحالات يمكن اللجوء إلى وسائل رياضيّة معقّدة، يتمّ بها تحويل الصورة إلي متّجه Vector يتكوّن من ٥ أو ٦ أرقام تصف الصورة وصفا كافيا، حيث يتمّ أخذ كميّة ضخمة من العينات من الصور، يتمّ التعامل مع المتجهات الناتجة عنها لاستخلاص الاستنتاجات وتحديد كيفية التعرّف على الشكل المطلوب.

هذه طريقة.. الطريقة الأخرى هي استخدام الفئات العصبيّة Neural Networks للتعرّف على الأشكال.

عامّة هذا فرع شيّق من البرمجة وإن كان معقّدا، وله عشرات المراجع المستقلّة، وما زالت به مناطق شاسعة قيد البحث والتطوير، فشمّر ساعدك لو أردت.

لهذا فلن نخوض في هذا الأمر ها هنا.. كل ما نريد إيضاحه هــو كيــف يمكنك التعامل مع نقاط الصورة.. ولن يفيدنا في شيء التوسع في الأمثلة حول معالجة الصور.

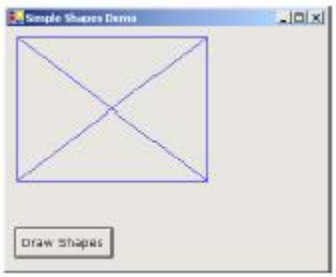
ولكن تعال أولا نرى استخداما هاما جدّا لكائن الصورة النقطيّــة، وذلــك باستخدامه في تثبيت الرسوم.

إبقاء الرسوم على النموذج والأدوات:

إنّ كلّ ما ترسمه على النموذج والأدوات يختفي لو صغّرت النموذج ثــمّ كبّرته، أو جعلت نموذجا آخر يخفي جزءا من نموذجــك ثــم أبعدتــه.. الصور فقط هي التي لا تُمحى إلا عندما تزيلها أنت.. بطريقة أخرى: إنّ الرسوم ــ على خلاف الصور ــ مؤقّتة، ولا تُضاف لكائن الرسوم.

ولحلّ هذه المشكلة، يمكنك أن تكتب كود الرسم فــي الحــدث Paint، أو الدالة التي تستدعيه OnPaint، بحيث يتمّ تنفيذه كلّمــا احتــاج النمــوذج لإعادة رسمه.

إنّ للنموذج إجراءً خاصّا Private يدعى OnPaint، يمكنك أن تســتبدله Override بإجراء آخر بنفس الاسم، لكي يُنفّذ كلّما احتاج النموذج لإعادة رسمه.. لقد كتبنا فيه الكود الذي يرسم الشكل التالي على النموذج:




```
Protected Overrides Sub OnPaint(
            ByVal e As PaintEventArgs)
    Dim G As Graphics
    G = e.Graphics
    Dim P As New Pen(Color.Blue)
    G.DrawRectangle(P, 10, 10, 200, 150)
    G.DrawLine(P, 10, 10, 210, 160)
    G.DrawLine(P, 210, 10, 10, 160)
End Sub
```

ولكنّ المشكلة لم تحلّ بعد!!.. فلو كانت الرسوم معقّدة وتستهلك وقتا، فسيبدو أداء برنامجك عقيما كلّما انتقلت من النموذج ثمّ عدت إليه.. ثمّ من قال إنّك ترسم دائما رسوما ثابتة؟.. إنّها قد تتغيّر مع ضغط المستخدم للقوائم والأزرار لاختيار وظائف معيّنة.. في هذه الحالة ستحتاج لكتابة كود معقّد في حدث رسم النموذج حتّى تستعيد الرسم كما كان.

إذن ما الحل؟

الحلّ هو أن نضيف الرسومات إلى كائن صورة نقطية، بحيث تصبح كالصور لا نحتاج لإعادة رسمها.

يا لها من فكرة!.. ولكن كيف ننفّذها؟

في البداية سنعرّف كائن صورة نقطيّة Bitmap Object، وهو مماثل لكائن الصورة Image Object، لدرجة أنّك تستطيع تحميل الصورة من أحدهما للآخر.

والآن سنعرّف كائن صورة نقطيّة له نفس مساحة مربّع الصورة:

```
Dim bmp As New Bitmap(
    PictureBox1.Width, PictureBox1.Height)
```

هذا الكائن يمثّل صورة فارغة الآن.

الخطوة الثانية هي أن نربط كائن الصورة الخاصّ بمربّع الصورة، بكائن الصورة النقطيّة.

```
PictureBox1.Image = bmp
```

وأنت تعرف طبعا أنّ هذا التساوي مرجعيّ ByRef.. الآن صار أيّ تغيير يحدث للكائن bmp، يحدث كذلك لكائن الصورة.

الخطوة الثالثة هي أن ننشئ كائن رسوم يرسم على الصورة النقطيّة، وذلك كالتالي:

```
Dim G As Graphics
G = Graphics.FromImage(bmp)
```

والآن، كلّ ما نرسمه بكائن الرسوم يتمّ رسمه على كائن الصورة النقطيّة bmp، وبالتالي يتأثّر به كائن الصورة Image الخاصّ بمربّع الصورة، وبالتالي يصبح رسما دائما في مربّع الصورة.

هذا هو الكود وقد جمعناه معا في دالّة واحدة، ليسهل استخدامها بعد ذلك:

```
Function GetGraphicsObject(
        ByVal PBox As PictureBox) As Graphics
    Dim bmp As Bitmap
    bmp = New Bitmap(PBox.Width, PBox.Height)
    PBox.Image = bmp
    Dim G As Graphics =
        Graphics.FromImage(bmp)
    Return G
End Function
```

والآن كلّ ما عليك هو إرسال اسم مربّع الصورة إلى هذه الدالة، لتعيد إليك كائن رسوم يرسم رسوما ثابتة على سطحه.

لاحظ أنّك لو أردت أن تفعل المثل مع النموذج، فستجد أنّه لا يمتلك خاصيّة Image.. لا عليك: استخدم خاصيّة "صورة الخلفيّة" BackgroundImage بدلا منها.. هذه هي الدالة التي تتعامل مع النموذج:

```
Function GetGraphicsObject( ) As Graphics
    Dim bmp As Bitmap
    bmp = New Bitmap(Me.Width, Me.Height)
    Dim G As Graphics
    Me.BackgroundImage = bmp
    G = Graphics.FromImage(bmp)
    Return G
End Function
```

انظر كيف نستخدم هاتين الدالتين في مشروع SimpleGraphics في مجلّد برامج هذا الفصل.

ملحوظة:

يمتلك النموذج والأدوات الوسيلة Invalidate، لمسح الأشكال غير الثابتة من عليها.. ولو ناديت هذه الوسيلة بدون معاملات فستؤثّر على كلّ مساحة النموذج أو الأداة.. ولو ناديتها وأرسلت لها مستطيلا كمعامل، فستؤثّر على مساحة النموذج أو الأداة، التي تقع داخل هذا المستطيل.. طبعا هذه الوسيلة لن تؤثّر على الرسوم التي أضفناها لصرة نقطيّة.

ملاحظات على تغيير شكل النموذج:

رأينا في الفصل الخامس "النماذج" كيف نغيّر شكل النموذج.. ولقد استخدمنا الكود التالي لجعل النموذج بيضاويّا:

```
Private Sub Form1_Paint(ByVal sender As Object,
        ByVal e As PaintEventArgs) _
        Handles MyBase.Paint
    Dim G As Drawing.Graphics =
        Me.CreateGraphics
    Me.TransparentKey = Color.Red
    G.Clear(Color.Red)
    G.FillEllipse(Brushes.LightGray, 0, 0, Me.Width,
        Me.Height)
End Sub
```

جميل.. ما رأيك أن نستخدم طريقة الرسم على صورة نقطيّة، بدلا من استخدام الحدث Paint؟

ضع الكود التالي في حدث تحميل النموذج:

```
Private Sub Form1_Load(ByVal sender As Object,
            ByVal e As System.EventArgs) _
            Handles MyBase.Load
        Dim Bt As New Bitmap(Me.Width, Me.Height)
        Dim G As Drawing.Graphics =
            Drawing.Graphics.FromImage(Bt)
        Me.TransparentKey = Color.Red
        G.Clear(Color.Red)
        G.FillEllipse(Brushes.LightGray, 0, 0, Me.Width,
            Me.Height)
        Me.BackgroundImage = Bt
End Sub
```

جميل.. لكن لو جرّبت هذا الكود فلن يعمل!!

نتيجة مدهشة وغير متوقّعة.. صحّ؟

يا ترى ما هو السبب؟

السبب هو أنّ مفتاح الشفافية يتعامل فقط مع سطح النموذج، لهذا فهو يتجاهل الصورة في هذه الحالة تماما!

إذن فقد فشلت هذه الوسيلة؟

لا تتعجّل النتائج.. حينما حدثت معي هذه المشكلة لم أستسلم بسهولة.. جربت الكثير من الحلول، حتّى هداني الله سبحانه إلى الحلّ التالي:

١- لوّن خلفيّة النموذج بلون مفتاح الشفافية (الأحمر في مثالنا هذا) في وقت التصميم.. وانتبه لتغيّر لون الزرّ بدوره.. أعده للونه الأصليّ.

٢- في بداية حدث التحميل، أجبر النموذج على الظهور:

```
Me.TransparentKey = Color.Red
Me.Show
```

بهذا سيتمّ حذف المساحة الشفافة (كلّ سطح النموذج في هذه الحالة).

٣- بعد هذا يمكن وضع الصورة البيضاويّة على النموذج، بحيث تشـــكل مساحة سطحه.. هذا مع إلغاء جملة:

G.Clear(Color.Red)

لأنّها تعمل على تغطية المساحة الشفافة بصورة حمراء، ممّـــا يلغـــي الشفافية!!

هذا هو الكود:

```
Private Sub Form1_Load(ByVal sender As Object,
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
    Me.TransparentKey = Color.Red
    Me.Show( )
    Dim Bt As New Bitmap(Me.Width, Me.Height)
    Dim G As Drawing.Graphics =
        Drawing.Graphics.FromImage(Bt)
    G.FillEllipse(Brushes.LightGray, 0, 0, Me.Width,
        Me.Height)
    Me.BackgroundImage = Bt
End Sub
```

ولو جرّبته لوجدته يعمل بإذن الله.. مبارك.

التعامل مع نقط الصورة في الذاكرة مباشرةً:

تتسم الوسيلتان GetPixel و SetPixel بنوع ملحوظ من البطء، حتّى إنّك ستلاحظه في عمليّة تافهة كعكس الألوان.. المشكلة أنّ التعامل مع الصور لا يكون بنفس بساطة برنامج عكس الألوان، فلو فحصت العمليّات الأخرى الموجودة في مشروع ImageProcessing، فستجد أن تغيير قيمة أيّ نقطة يستلزم إجراء بعض العمليّات الحسابيّة على النقاط الثمانية المحيطة بها من الجهات الأربع (وأحيانا تمتدّ العمليات إلى غير ذلك من النقاط).. في هذه الحالة سيكون من العبث استدعاء الوسيلة GetPixel تسع مرّات لتغيير كل نقطة!

وأبسط تفكير يردُ على الخاطر في هذه الحالة، هو قراءة كل نقاط الصورة مرّة واحدة وحفظها في مصفوفة، والتعامل بعد ذلك مع المصفوفة عند الحاجة لقراءة أيّ نقطة.

ولكنّ هذا لن يحلّ مشكلة البطء نهائيّا، كما أنّه يستهلك الذاكرة!

لهذا فإنّ لدينا حلاّ آخر.. ما رأيك لو تعاملنا مع النقاط في أماكن تخزينها في الذاكرة مباشرة؟

إنّ ذلك سيكون صعبا نوعا ما، وفيه بعض المخاطرة، فمن الممكن أن تؤدّى إلى إغلاق البرنامج أو الويندوز كلّه، لو كتبت في منطقة خاطئة من الذاكرة، فأفسدت بعض بيانات البرامج الأخرى أو النظام.

توكّل على الله وهيّا اتبع معي هذه الخطوات:

في البداية تعال نختار الجزء الذي نريد التعامل معه من الصورة، ونضعه في كائن بيانات الصورة النقطيّة BitmapData.. ولاستخدام هذا الكائن، لا تنسَ أن تكتب جملة الاستيراد التالية في مشروعك:

Imports System.Drawing.Imaging

عرّف كائن صورة نقطيّة وضع فيه صورة مربّع النصّ:

```
Dim Bitmap As New Bitmap(PictureBox1.Image)
```

بعد ذلك عرّف مستطيلا له نفس مساحة الصورة:

```
Dim Rect As New Rectangle(0, 0, Bitmap.Width,
    Bitmap.Height)
```

ثمّ عرّف كائن بيانات الصورة النقطيّة كالتالي:

```
Dim BitmapData As New BitmapData( )
```

ثمّ ضع الصورة في كائن بيانات الصورة النقطيّة باستخدام الوسيلة "إغلاق خانات الصورة" LockBits:

```
BitmapData = Bitmap.LockBits(Rect,
    ImageLockMode.WriteOnly, Bitmap.PixelFormat)
```

هذه الوسيلة تمنع تغيير موضع الصورة في الذاكرة (بفعل جامع القمامة GC أو الويندوز) إلى حين الانتهاء من التعامل معها.. وهي تأخذ ثلاثة معاملات: أوّلها مستطيل يحدّد المساحة التي ستمنع التعامل معها من الصورة، والثاني نوع الإغلاق، وقد اخترنا منع الكتابة على الصورة، والثالث هو نوع تنسيق الصورة.

والآن صار كائن بيانات الصورة يشير إلى بيانات الصورة في الذاكرة، ولكي نحصل على عنوان أوّل نقطة في الصورة في الذاكرة، سنستخدم الوسيلة Scan0.. حيث سنقوم بحفظ العنوان في متغيّر مخصّص لهذا النوع من البيانات، هو المؤشّر Pointer من النوع مؤشّر صحيح IntPtr:

```
Dim pixels As IntPtr = BitmapData.Scan0( )
```

طبعا هذه أوّل مرّة يتعامل فيها مبرمج VB مع المؤشّرات مباشرة (بدون دوال API).. ولا بدّ أنّ التساؤل الذي يحاصره هو: وماذا سنفعل بالمؤشّر و VB لا يتعامل مع الذاكرة مباشرة؟

لقد كان ذلك في الماضي، أمّا الآن، فإنّ لـديك فئـة تسـمّى Marshal، تمكّنك من القراءة من الذاكرة والكتابة فيها، عن طريق المؤشّرات.. ابـدأ بتعريف متغيّر من هذه الفئة كالتالي:

Dim M As System.Runtime.InteropServices.Marshal

وهي تمنحك عشرات الوسائل للتعامل مع الذاكرة، ولكنّ ما يهمّنا هنا هـو القراءة والكتابة في الذاكرة.. وفي هذا الصدد لدينا العديد مـن الوسـائل المتماثلة، التي لا تختلف إلا في عدد الوحدات التي تقرأها أوتكتبهـا فـي الذاكرة.. فللقراءة يمكنك استخدام:

اقرأ وحدة **ReadByte** ــ اقرأ عدد صـحيح قصـير **ReadInt16** ــ اقـرأ عـدد صـحيح **ReadInt32** ــ اقـرأ عـدد صـحيح طويـل **ReadInt64**.

وكلّ هذه الوسائل تأخذ معاملين: المؤشّر الذي يشير لموضـع الـذاكرة، وعدد الوحدات التي تريد إضافتها لهذا المؤشّر Offset، حتّى تتمكّن مـن قراءة المواضع التالية له.. وطبعا تعيد كلّ وسيلة نفس نوع البيانات الذي تقرأه.

وللكتابة في الذاكرة استخدم الوسائل:

اكتب وحدة **WriteByte** ــ اكتب عدد صحيح قصير **WriteInt16** ــ اكتب عدد صـحيح **WriteInt32** ــ اكتـب عـدد صـحيح طويـل **WriteInt64**.

وتأخذ هذه الوسائل نفس معاملي وسائل القراءة، بالإضافة لمعامل ثالـث، هو القيمة التي تريد أن تكتبها في الذاكرة، مع مراعاة أن تكون من نفـس النوع الذي تكتبه الوسيلة.

بقي شيء هامّ، هو معرفة كيف تحفظ النقاط في الذاكرة:

إنّ كل نقطة تأخذ عددا من وحدات الذاكرة Bytes يختلف على حسب نوع تنسيق الصورة.. لقد افترضنا هنا أنّ الصورة ملوّنة، وأنّ كلّ نقطة من نقاطها تخزّن في أربع وحدات ذاكرة.

ويجب أن تعرف، أنّ التعامل مع الذاكرة سيختلف عن التعامل مع الصورة، فقد كنا نحصل على النقطة بمعلوميّة الصف والعمود.. ليس في الذاكرة هذا التقسيم، وإنّما يتمّ حفظ نقاط الصورة على التوالي: توضع نقاط الصفّ الأوّل، ثمّ يليها نقاط الصف الثاني، فالثالث وهكذا...

افترض أنّ هذه هي خانات الصورة (سنكتب في كلّ خانة رقمها للإيضاح):

0	1	2	3	4
5	6	7	8	9

هكذا تخزّن في الذاكرة:

0	1	2	3	4	5	6	7	8	9

فإذا علمت أنّ كل نقطة تخزّن في ٤ وحدات ذاكرة، فإنّ هذا يعني أنّ أوّل نقطة في الصف الثاني من الصورة (لاحظ أنّ رقمه ١، لأنّ الصفّ الأوّل رقمه صفر) ستوضع في خانة الذاكرة التي تساوي: عرض الصورة (٥ خانات في مثالنا هذا) × ٤، والتي تليها (لاحظ أنّ رقم عمودها ١، لأنّ العمود الأوّل رقمه صفر) ستكون في الخانة رقم: عرض الصورة × ٤ + ١ × ٤، وهكذا .. وبصورة عامّة:

الخانة رقم ع في الصفّ الثاني ستوضع في خانة الذاكرة رقم:

٤ × (عرض الصورة + ع).

كما أنّ أوّل خانة في الصفّ الثالث ستوضع في الخانة رقم: ٤ × (٢ × عرض الصورة)، والتي تليها ستكون في الخانة: ٤ × (٢ × عرض الصورة + ١).

وبصورة عامّة: لمعرفة موضع نقطة في الذاكرة، تقع في الصورة في الصفّ ص والعمود ع، طبّق المعادلة:

رقم الخانة في الذاكرة = (ص × عرض الصورة + ع) × عدد الوحدات التي تمثّل النقطة (٤).

والآن تعال نكتب دالّة عكس الألوان بهذه الطريقة:

```
Dim X, Y, C, O As Integer
Dim W As Integer = Rect.Width
Dim Clr As Color
For Y = 0 To Rect.Height - 1
    For X = 0 To W – 1
        ' حساب موضع النقطة في الذاكرة
        O = CInt(Y * W * 4 + X * 4)
        ' قراءة عدد صحيح من الذاكرة يمثّل اللون
        C = M.ReadInt32(pixels, O)
        ' تحويل اللون من عدد صحيح إلى كائن لون
        Clr = Color.FromArgb(C)
        ' عكس مكوّنات اللون وتكوين اللون المعكوس
        Clr = Color.FromArgb(Clr.A, 255 - Clr.R, _
            255 - Clr.G, 255 - Clr.B)
        ' تحويل كائن اللون إلى عدد صحيح
        C = Clr.ToArgb
```

```
                تخزين اللون المعكوس في الذاكرة '
                M.WriteInt32(pixels, O, C)
            Next
        Next
```

ولا تنس في النهاية تحرير المساحة التي أغلقناها في الذاكرة، مع وضع كائن الصورة النقطيّة في مربّع الصورة وإنعاشه، حتّى ترى التغيير:

```
Bitmap.UnlockBits(BitmapData)
PictureBox1.Image = Bitmap
PictureBox1.Refresh( )
```

ستجد هذا الكود كاملا في الأمر Inverse1 تحت القائمة Processفي مشروع ImageProcessing.. حاول أن تقارن بين هذه الطريقة وبين الطريقة التقليديّة التي كتبنا بها الأمر Inverse في نفس القائمة.. ستجد الفارق في السرعة واضحا (وصل معي إلى خمسة أضعاف)، وسيتّضح أكثر، لو كتبت العمليّات الأربع الأخرى الموجودة في نفس القائمة بالطريقة الجديدة، فهي بالفعل تأخذ وقتا شنيعا!

ولو شئت تعديل كود عكس الألوان ليتعامل مع صور بتنسيقات مختلفة، تُخزَّن فيها النقطة في وحدة Byte أو اثنتين أو ثلاثة أو أربعة، فاستخدم الوسيلة "قراءة حجم تنسيق النقطة" GetPixelFormatSize، حيث يمكنك أن تعرف بها عدد الوحدات التي تخزّن فيها كلّ نقطة كالتالي:

```
N = Bitmap.GetPixelFormatSize(Bitmap.PixelFormat)
```

وبناء على هذا الرقم، عدّل الكود ليقرأ N من وحدات الذاكرة في كلّ مرّة بدلا من ٤.. لن يكون ذلك بسيطا، ولكنّه أيضا لن يكون معقّدا.. اعتبره تدريبا لك.. هيّا يا بطل!

تغيير لون معيّن في الصورة إلى لون آخر:

لحسن الحظّ، لن تحتاج إلى المرور على كل نقاط الصورة لونا لونا، بحثا عن اللون الذي تريد تغييره.. إنّ لدينا طريقةً جاهزةً لفعل ذلك.. اتّبع الخطوات التالية:

ابدأ بتعريف كائن الصورة النقطيّة، وضع به الصورة التي تريد تغييرها (ولتكن تلك الموجودة في مربّع الصورة):

```
Dim image As New Bitmap(PictureBox1.Image)
```

الآن سنعرّف كائنا يسمّى خريطة اللون ColorMap.. لهذا الكائن خاصيتان هامّتان:

اللون القديم OldColor:

وهو الذي تريد إزالته من الصورة.

اللون الجديد NewColor:

وهو الذي تريد استبداله باللون القديم.

وبإمكانك أن تعرّف مصفوفة من هذا الكائن، بحيث تستبدل مجموعة من الألوان بغيرها مرّة واحدة.. وفي مثالنا هذا سنعرّف مصفوفة من خانة واحدة، نستبدل بها اللون الأزرق (اللون الجديد) باللون الأحمر (اللون القديم):

```
Dim colorMap( ) As ColorMap = {New ColorMap ( )}
colorMap(0).OldColor = Color.Red
colorMap(0).NewColor = Color.Blue
```

ثمّ عرّف كائنا من النوع سمات الصورة ImageAttributes:

```
Dim imageAttributes As New ImageAttributes( )
```

الآن سنستخدم الخاصيّة "تغييـــر خريطـــة الألـــوان" SetRemapTable الخاصّة بكائن سمات الصورة لجعل مصفوفة خريطة التغيير جزءا مـــن سمات الصورة:

imageAttributes.SetRemapTable(colorMap)

والآن لم يبقَ إلا أن تعيد رسم الصورة، حتّـــى تبـــدو التعـــديلات بهـــا.. وسنستخدم في هذا وسيلة رسم الصور DrawImage الخاصّـــة بكـــائن الرسوم.. ومن الطبيعيّ أن نرسم بها على الصورة النقطيّة، حتّـــى تظـــلّ الصورة المرسومة ثابتة كما شرحنا من قبل:

```
Dim W As Integer = PictureBox1.Width
Dim H As Integer = PictureBox1.Height
Dim G As Graphics
' إنشاء كائن رسوم من الصورة النقطيّة
G = G.FromImage(image)
' الرسم على الصورة النقطية بالسمات التي ضبطناها، وبذلك نغيّر اللون الأحمر بالأزرق
G.DrawImage(image, New Rectangle(0, 0, W, H), _
0, 0, W, H, GraphicsUnit.Pixel, imageAttributes)
' وضع الصورة في مربّع الصورة
PictureBox1.Image = image
' إنعاش مربّع الصورة
PictureBox1.Refresh( )
```

هل تبدو صيغة وسيلة رسم الصورة DrawImage جديدة عليـــك؟.. لا عليك، فإنّ لها ٣٠ صيغة مختلفة!

وهذه هي المعاملات بالترتيب:

- الصورة التي سنرسمها.
- المستطيل الذي يحدّد المساحة التي سنرسم فيها.

- المعاملات الأربعة التالية تحدّد المساحة التي سنرسمها من الصورة، وهي بالترتيب: الإحداثيين الأفقي والرأسي لزاوية المستطيل العليا اليسرى، وعرض المستطيل، وارتفاعه.
- وحدة قياس الأطوال، وفي حالتنا هذه هي النقطة Pixel.
- وأخيرا المعامل الذي يهمّنا والذي فعلنا كلّ ذلك من أجله: سمات الصورة، حيث أرسلنا كائن سمات الصورة الذي يحمل خريطة تغيير الألوان.

ويمكنك أن تجرّب هذا المثال في مشروع ImageProcessing، في الأمر ChangeColor في قائمة Processing، بشرط أن تضع في مربّع الصورة صورة بها مساحة حمراء، حتّى تشعر بحدوث اختلاف.

لا بدّ أنّه قد ساءك أنّ المستخدم مجبر على تغيير اللون الأحمر إلى الأزرق.. فماذا لو أراد أن يغيّر أيّ لون في الصورة إلى أيّ لون آخر؟

كتدريب خاصّ، حاول تعديل الكود الخاص باستبدال الألوان، ليسمح للمستخدم باختيار اللون الذي يريد تغييره، واختيار اللون البديل، وذلك باستخدام الأداة ColorDialog.

وعلى فكرة: يمكنك تغيير أيّ لون ليصير شفافا Color.Transrarent، أو ليصير أي لون به نسبة شفافية (باستخدام معامل الشفافية في دال تكوين اللون Color.FromARGB).. هذه نقطة مفيدة جدّا.

تغيير نقاط الصورة:

لكي تتمكّن من التعامل مع كلّ نقطة في الصورة على حدة، يمنحك كائن الصورة النقطيّة الوسيلتين التاليتين:

اقرأ النقطة GetPixel:

تأخذ هذه الوسيلة إحداثيي النقطة المطلوبة، وتعيد لك لونها:

```
Dim Clr As Color = Bitmap.GetPixel(X, Y)
```

وتستهلك هذه الوسيلة وقتا ملموسا، خاصّة مع تكرار استدعائها لكلّ نقاط الصورة.

تغيير النقطة SetPixel:

تأخذ هذه الوسيلة إحداثيي النقطة المطلوبة واللون الذي تريد تلوينها به:

```
Bitmap.SetPixel(X, Y, Clr)
```

وفي المثال التالي نرى كيف يمكننا عكس ألوان صورة ما للحصول على النيجاتيف منها.

لقد كتبنا في هذا الفصل دالة عكس الألوان RevColor.. والآن كلّ ما سنفعله، هو أن نطبّقها على كلّ نقطة في الصورة لعكس لونها.

ولقراءة نقاط الصورة وتغييرها، لا بد من كتابة جملتين تكراريّتين متداخلتين، إحداهما لقراءة الصفوف والأخرى لقراءة الأعمدة.

هذا هو الكود الذي يعكس الألوان:

```
Dim R, C As Integer
' تعريف متغير صورة نقطيّة، وتحميل صورة مربّع الصورة فيه
Dim Btmap As New Bitmap(PictureBox1.Image)
' ربط مربّع الصورة بكائن الصورة النقطيّة، حتّى يتأثّر بالتغيرات التي تحدث له
```

```
PictureBox1.Image = Btmap
' جملة تكراريّة للمرور عبر صفوف الصورة
For R = 3 To Btmap.Width - 3
    ' جملة تكراريّة للمرور عبر نقاط كلّ صفّ
    For C = 0 To Btmap.Height - 3
        Btmap.SetPixel(R, C,
            RevColor( Btmap.GetPixel(R, C) ) )
    Next
Next
PictureBox1.Refresh( )
```

ويقدّم لك مشروع ImageProcessing أربـــع عمليّـــات أخـــرى علـــى الصور.. حاول تجربته.

رسم الدوالّ Plotting Functions:

سنختتم هذا الفصل بهذا الموضوع، الذي ستجده شيّقا بإذن الله.. ولمن لا يحبّون الرياضيّات، يمكنهم ألا يكتئبوا، فهذا الجزء بمثابة مشروع تدريبيّ ليس أكثر، وإن كنت أنصحهم بقراءته على كلّ حال، فسنتعرّف فيه على أشياء جديدة، كما سنتعامل فيه مع أداة هامّة، قد تفيدهم فيما بعد.

فلنبدأ:

افترض أنّ لديك المعادلة: ص = ٢س.

والآن لرسم هذه الدالة، لا بدّ أن نوجد قيم ص المناظرة لكلّ قيم س (٠، ١، ٢، ٣، ... إلخ).

ثم نقوم برسم النقاط الناتجة، حيث إنّ كلّ نقطة هي عبارة عن الإحداثيين (س، ص).. فمثلا في دالتنا هذه ستكون النقطة على الصيغة (س، ٢س).. مثل: (٠، ٠) ، (١، ٢) ، (٢، ٤) ... إلخ.

ولا بدّ أن نصل بين هذه النقاط بخطوط مستقيمة، حتّى يبدو المنحنى متصلا.

وهنا ستبرز هذه المشاكل:

١- إذا كان معدّل تغيّر الدالة كبيرا، فستجد أنّ النقاط متباعدة، بحيث ستكون الخطوط الواصلة بينها واضحة للعيان، وسيبدو المنحنى متكسرا غير ناعم.. لحلّ هذه المشكلة نحتاج لتغيير مقياس رسم المحور السيني (الأفقيّ).. وللقيام بذلك نطبّق المعادلة التالية:

مقياس الرسم الأفقي = عرض مربّع الصورة ÷ (أكبر س – أصغر س).

فمثلا: لو كنا سنرسم الدالة لقيم س بين -١ و ٥، فسيكون:

مقياس الرسم = عرض مربّع الصورة ÷ (٥ – [-١])

= عرض مربّع الصورة / ٦.

٢- إذا كانت قيم ص الناتجة من الدالة كبيرة، فإنّ كثيرا منها لن يظهر في منطقة الرسم.. ولحلّ هذه المشكلة، نحتاج لتغيير مقياس رسم المحور الصادي (الرأسيّ).

مقياس الرسم الرأسيّ = ارتفاع مربّع الصورة ÷ (أكبر ص – أصغر ص).

٣- إذا كانت هناك قيم سالبة على المحور السينيّ أو الصاديّ، فإنّها لن تظهر في مساحة الرسم (لأنّ إحداثيات النموذج والأدوات تبدأ من (٠، ٠).. ولحلّ هذه المشكلة، يجب نقل الإحداثيّات، حتّى تصبح أكبر قيمة سالبة منطبقةً على الصفر.

وفي حالتنا هذه، سنقوم برسم النقط في كائن مسار الرسوم GraphicsPath، حيث سنصل بخطّ مستقيم، بين كلّ نقطة جديدة نضيفها إليه والنقطة السابقة لها، ثمّ سنرسم كائن المسار مرّة واحدة.. إنّ هذه الطريقة أسرع من رسم كلّ نقطة على حدة، كما أنّها تمكّنك من استخدام المتغيّر الذي يشير لكائن مسار الرسوم لإعادة رسم الدالة أكثر من مرّة بمقاييس رسم مختلفة.

ولكن كيف نقوم بهذه التحويلات؟

قلنا إنّ عمليات التحويل يتّم حفظها في مصفوفة التحويل.. لهذا سنستخدم هذه المصفوفة في حالتنا هذه، وسنجري عليها التحويلات مباشرة، وأوّل ما سنفعله هو تعريف متغيّر من مصفوفة التحويل:

```
World = New System.Drawing.Drawing2D.Matrix( )
```

ملحوظة:

هذه المصفوفة تتكوّن من ٣ صفوف و ٣ أعمدة، وهي تمتلك وسائل جاهزة لتدوير المصفوفة Rotate وعكسها Invert وضربها في مصفوفة أخرى Multiply.. ولقراءة محتويات مصفوفة التحويلات والحصول على مصفوفة عاديّة Array ٣×٣ استخدم الخاصيّة Elements.

بعد ذلك سنغيّر مقياس الرسم أفقيّا ورأسيّا:

```
World.Scale(((PictureBox1.Width - 4) / (Xmax - Xmin)), _
-(PictureBox1.Height – 4) / (Ymax - Ymin))
```

لاحظ أنّنا طرحنا ٤ من عرض وارتفاع مربّع الصورة، حتّى نزيل عرض الحوافّ ونترك هامشا بينه وبينها.. لاحظ أيضا أن أصغر وأكبر قيمتين على المحور الأفقي Xmin و Xmax هما اختياريتان، على حسب المدى الذي تريد رسم الدالّة فيه.. أمّا أصغر وأكبر قيمة على المحور الرأسي فلا بدّ من حسابهما أولا، سواء بالوسائل الرياضيّة التي تحسب القمّة العظمى والصغرى (سيحتاج هذا لإجراء عمليات تفاضل، وهذا خارج نطاقنا الآن)، أو بجملة تكراريّة تحسب كل قيم ص المناظرة لقيم س، بحيث تلتقط من بينها أصغر قيمة وأكبر قيمة.

بعد ذلك سنحرّك الإحداثيّات لتظهر كلّ النقط في منطقة الرسم:

```
World.Translate(-Xmin, -Ymax)
```

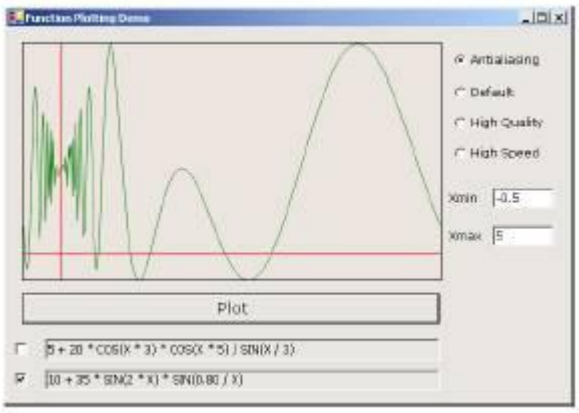
لاحظ أنّ الإحداثيّات في الكتب العلميّة والرياضيّة تفترض أنّ القيم على المحور الرأسي تزداد لأعلى وتقلّ لأسفل، لهذا فقد نقلنا الإحداثيّ الرأسيّ لأسفل بمقار أكبر قيمة له، بحيث تظهر هذه القيمة في أعلى الشاشة (ارتفاعها = ٠).

الآن أصبحت مصفوفة التحويلات تحتوي على كل التحويلات المطلوبة، ولكي تنطبق هذه التحويلات على كائن مسار الرسوم، فإنّنا نستدعي الوسيلة Transform الخاصّة به، ونرسل هذه المصفوفة كمعامل لها، ثمّ بعد ذلك نرسمه:

GraphicPath1.Transform(World)
G.DrawPath(plotPen, plot1)

أعتقد الآن أن التساؤل الذي دار في ذهنك قد أجيب: إنّ الفارق بين استخدام مصفوفة التحويلات ووسائل التحويل الخاصّة بكائن الرسوم Graphics، هي أنّ مصفوفة التحويلات خاصّة فقط بالكائن الذي يستخدمها (كائن مسار الرسوم هنا)، أمّا وسائل التحويل الخاصّة بكائن الرسوم فهي عامّة لكل منطقة الرسم.

ولديك في مجلد برامج هذا الفصل، المشروع Plotting، وفيه سترى كيف يمكن رسم دالتين مختلفتين بواسطة المفاهيم التي شرحناها هنا.



أمّا لو أردت أن تسمح للمستخدم بتعريف الدالة التي يريد رسمها (بالصيغة الإنجليزيّة)، يمكنك استخدام الأداة Script ActiveX control.. ولإضافة هذه الأداة لبرنامجك، اضغط بزر الفأرة الأيمن على كلمة المراجع References في متصفّح المشاريع، ومن القائمة

الموضعيّة اختر Add Reference، وفي مربّع الحوار الـذي سـيظهر اضغط الشريط COM.. ومن بين العناصر التي تملأ القائمة انقر مرتين على العنصر Microsoft Script Control 1.0.. بعد ذلك أغلق مربّع الحوار.

الآن يمكنك كتابة دالة تحسب لك قيمة الصيغة التي عرّفها المستخدم عنـد قيمة معيّنة لـ X، باستخدام إحدى نسـخ VB المخصّصـة لصـفحات الإنترنت، وهي لغة VBScript، كالتالي:

```
Function FVal(X As Double, Formula As String) As Double
    Dim S As New MSScriptControl.ScriptControl
    S.Language = "VBScript"
    Try
        S.ExecuteStatement("X=" & X)
        Return  CDbl(S.Eval(Formula))
    Catch exc As Exception
        Throw New Exception("X=" & X & "لا يمكن حساب الدالة عند")
    End Try
End Function
```

والآن جرّب استدعاء هذه الدالة كالتالي:

```
Dim F As String
F = "5 + 20 * Cos(X * 3) * Cos(X * 5) / Log(Abs(Sin(X)) / 10)"
MsgBox(FVal(3, F))
```

عنـد تنفيـذ هـذا الكـود، سـتظهر لـك رسـالة تجمـل الـرقم: ١,٧٥٠٩١٠٣٢٧٣٠١٢٦.

جرّب وضع مربّع نصّ علـى نمـوذج، واكتـب فيـه صـيغة الدالـة (بالإنجليزيّة) ومربّع نصّ آخر تكتب فيه قيمة X التي تريـد أن تعـرف قيمة الدالة عندها، وزرا اكتب فيه ما يلي:

```
MsgBox(FVal(CDbl(TextBox2.Text), TextBox1.Text))
```

وفي ضوء هذا، حاول أن تطوّر مشروع رسم الدوال Plotting، ليسمح للمستخدم بتعريف صيغة الدالة التي يريدها.

عامّة ستجد هذا متحقّقا بالفعل في المشروع FunctionPlotting.

ولو أرد تدريبا جيّدا، فعليك أن تكتب دالة تحوّل الصيغ العربيّة إلى الصيغ الإنجليزيّة، مثل:

÷	/
×	*
جا	Sin
جتا	Cos
|.....|	Abs(.....)
لوج	Log

ويفضّل أن تستخدم في هذا محرّك "التعبيرات النمطيّة" Regular Expression الذي يمنحه لك إطار العمل.. (مشروح بالتفصيل في مرجع "**من الصفر إلى الاحتراف: برمجة إطار العمل**").

الفصل الثاني

الأدوات الخاصّة
Custom Controls

القوّة.. والسهولة.. ومنتهى الروعة:

لا ريب أنّك ستستمع جدّا بهذا الفصل، ففيه ستتعلّم كيف تنشئ الأدوات Controls التي تروق لك، بحيث توفّر على نفسك وقت إعادة تصميم النماذج التي تستعملها كثيرا.

وكما هي القاعدة في VB.Net، يجب أن تتوقّع أنّك تتعامل مع أسهل لغة برمجةٍ معروفة، بخلاف القوّة الجديدة التي اكتسبتها مؤخّرا.. باختصار: يمكنك أن تُنشئ بهذه اللغة أدوات رائعة عالية الكفاءة.

ويكفي أن أخبرك أنّك تستطيع أن ترث أيّ أداة من أدوات اللغة، لتعدّل فيها أو تضيف بعض الخصائص والوسائل إليها.. إنّ هذا سيضمن لك مبدئيّا أنّك لن تبدأ من الصفر في كلّ مرّة!

أمّا ما سيثلج صدرك تماما، فهو أنّك تستطيع الآن إنشاء أدوات جديدة حتّى في شكلها، دون أن ترتبط بالأشكال التقليديّة للأدوات (خاصة الأشكال المستطيلة المستفزّة)، وذلك عن طريق استخدام دوال الرسم والتلوين لرسم الأداة كما يحلو لك، بل ويصل الأمر إلى قدرتك على تغيير شكل قوائم النموذج Menus!

أعتقد أنّي قد أثرت فضولك بما يكفي، (لدرجة أنّني أعتقد أنّ هذا المكان مناسب لوضع إعلان عن أيّ منتج تافه على غرار ما تفعله المسلسلات!!!)، فتعال نبدأ بدون تضييع لحظة واحدة زائدة.

تطوير الأدوات الموجودة

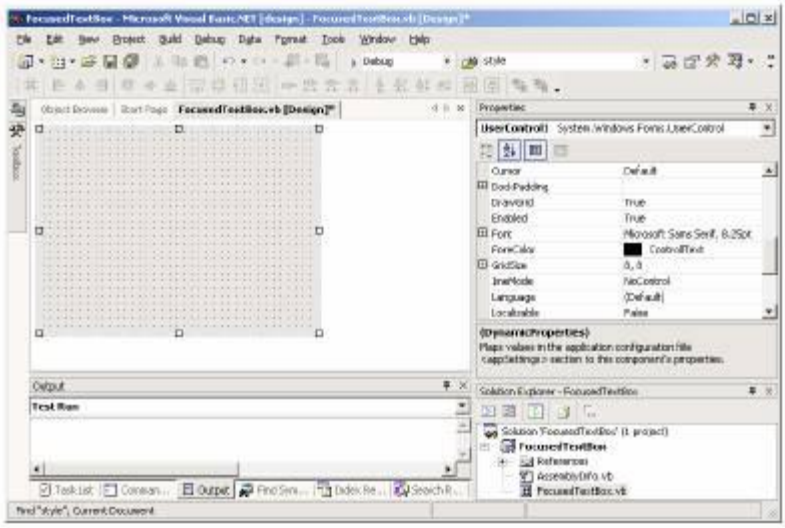
أبسط طريقة لبناء الأدوات، هي وراثة أداة موجودة وتطويرها.
افترض أنّك تريد أن تصمم نموذجا يستقبل البيانات من المستخدم عن طريق مجموعة من مربّعات النصّ.. وافترض كذلك أنّك تريد أن نمنع المستخدم من كتابة أيّ حروف في مربّعات النصّ، بحيث يقتصر فقط على كتابة الأرقام.. في هذه الحالة ستضطرّ لتكرار كتابة الكود الذي يمنع المستخدم من كتابة الحروف في كلّ مربّع نصّ لديك.. الحلّ الأفضل هو أن تنشئ أداة جديدة ترث مربّع النصّ، وتحتوي على خاصيّة تحدّد نوعيّة البيانات التي يقبلها مربّع النصّ: نصوص، أرقام، أعداد عشريّة، لا يقبل علامات ترقيم... إلخ.. إنّ هذه الأداة ستكون مفيدة لك كثيرا في تطبيقات إدخال البيانات.
تعال نرى:

تصميم الأدوات:

إن تصميم الأداة Control لن يكون جديدًا عليك كلّيّةً، فهي تتكوّن من شقّين:

١- واجهة الأداة:

ويمكنك تصميم هذه الواجهة كما تصمّم أيّ نموذج عاديّ، حيث تمنحك اللغة كائنا شبيها تماما بالنموذج، هو "أداة المستخدم" UserControl، التي يمكنك أن تضع عليها الأدوات أو الرسوم، بنفس الطريقة التي تفعل بها هذا مع النموذج.



٢- كود الأداة:

لا بدّ أن تقوم أداتك بوظيفة أو أكثر، كما لا بدّ لها أن تمنح من سيستخدمها العديد من الخصائص والوسائل والأحداث التي سيبرمجها بها، تماما كما تفعل الأدوات التقليديّة.. لا مشكلة: ستطبّق نفس القواعد التي تعلّمتها في إنشاء الفئات Classes، لإنشاء الخصائص والوسائل والأحداث اللازمة للأداة.. وحتمٌ عليّ أن أخبرك أنّ العديد من هذه الخصائص والوسائل والأحداث، تمنحها لك اللغة تلقائيّا دونَ أن تكتبَ أنت أيّ كود.

إذن يمكنك أن تقول باختصار: إنّ الأداة هي فئة Class لها واجهة استخدام (نموذج).

ولكنّ هناك بعض الفروق الهامّة بينَ الأداة والنموذج، منها أنّ الأداة لا يمكن أن تستخدم بمفردها، بل يجب أن توضع على نموذج أولا ليتمّ عرضها من خلاله.. ومنها أنّ برمجة الأداة يجب أن تراعي حالتي التشغيل التاليتين للأداة:

١- حالة استخدام المستخدم للأداة في وقت التشغيل.
٢- حالة استخدام المبرمج للأداة في وقت التصميم، بوضعها على نموذج وتعديل شكلها وخصائصها، فأنت المسئول أيضا عن سلوك الأداة في هذه الحالة، فرغم أنّه وقت تصميم للمبرمج، إلا إنّه وقت تشغيل للأداة!

من أجل هذا فإنّ أداة المستخدم UserControl تمنحك الخاصيّة "طور التصميم" DesignMode.. هذه الخاصيّة ترجع القيمة True إذا كانت الأداة في وقت التصميم، وترجع القيمة False إذا كانت في وقت التشغيل.. وبهذا تستطيع أن تحدّد ردّ الفعل المناسب الذي تستجيب به لأحداث المبرمج أو المستخدم.. لن نتطرّق لهذه النقطة مرّة أخرى في هذا الفصل، ولكن عليك أن تضعها في ذهنك، فستحتاجها حتما.

إنشاء مربّع النصّ الفعّال FocusedTextBox:

في هذا المثال سنبدأ بتطوير مربّع النصّ، بحيث يتغيّر لونه عندما يستقبل مؤشّر الكتابة، ويستعيد لونه الأصليّ عندما يفقد مؤشّر الكتابة.

ابدأ مشروعا جديدا، وفي مربّع حوار "مشروع جديد" اختر النوع "مكتبة أدوات ويندوز" Windows Control Library.. سمِّ المشروعَ FocusedTextBox واضغط موافق.. ستجد أنّ المشروع سيحتوي على أداة شبيهة بالنموذج، ولكن ليس لها أيّ حدود أو شريط عنوان.. هذه الأداة هي أداة المستخدم UserControl1.. هذه هي الأرضيّة التي ستصمّم عليها واجهة الأداة.

افتح نافذة الخصائص، وغيّر اسم أداة المستخدم إلى FocusedTextBox.. الآن علينا أن نجعل هذه الأداة ترث مربّع

النصّ.. بسيطة.. اضغط بزر الفأرة الأيمن علـــى أداة المســـتخدم ومـــن القائمة الموضعيّة اختَرْ "عرض الكود" View Code.. سيظهر لك الفئـــة التي تمثّل الأداة، وستجد فيها الكود التالي:

```
Public Class FocusedTextBox
    Inherits System.Windows.Forms.UserControl
End Class
```

تلاحظ أنّ سطر الوراثة ينصّ على أنّ هذه الأداة ترث فئة أداة المستخدم UserControl.. لا نريد ذلك.. قم بتغيير هذا السطر لترث مربّع النصّ كالتالي:

```
Inherits TextBox
```

الآن احفظ المشروع، وعد إلى تصميم الأداة.. ستكتشف أنّ أداة المستخدم قد اختفت، وظهرت مكانها مساحة فارغة.

ملحوظة:

إذا كانت أداة المستخدم ما زالت معروضة، فأغلق واجهة التصميم وأعد فتحها مرّة أخرى، وستجد التغيير المشار إليه قد حدث.

ولاختبار هذه الأداة يجب عليك إضافتها إلى نمـــوذج.. لهـــذا سنضـــيف مشروعا جديدا إلى التطبيق سنستخدمه لاختبار الأداة أثنـــاء تصـــميمها.. اضغط القائمة File\Add Project\New Project.. وفي مربّع حـــوار "مشروع جديد" اختر النوع Windows Application وسمّ المشـــروع TestProject.

الآن يمكنك اختبار الأداة بوضع نسخة منها على النمـــوذج Form1 فـــي مشروع الاختبار.. ولفعل ذلك اتبع هذه الخطوات اليسيرة:

- حـــدد العنصـــر FocusedTextBox فـــي متصـــفّح المشـــاريع Solution Explorer، ومن القائمة "بناء" Build اضـــغط الأمـــر

Build FocusedTextBox، وذلك لبناء ملف Dll لـــلأداة يمكـــن استخدامه.

- حدد العنصر TestProject في متصفّح المشاريع، واجعـــل هـــذا المشروع هو مشروع بدء التشغيل، بضغط الأمر Set As Startup Project في قائمة Project.
- افتح النموذج Form1.
- فـــــي صـــــندوق الأدولـــــت ToolBox ســـــتجد اســـــم الأداة FocusedTextBox موجودا (آخر أيقونة).. أضف نسخة من هذه الأداة للنموذج.
- الآن حاول استخدام هذه الأداة.. ستكتشف أنّك تتعامل مـــع مربّـــع النصّ التقليديّ بدون أيّ اختلاف.

الآن تعال نضيف بعض الوظائف الجديدة لمربّع النصّ.. عُد إلى الأداة، وفي محرّر الكود Code Editor اضغط القائمة المنسدلة اليسرى، واختر منها العنصر "أحداث الفئة الأساسيّة" Base Class Events.. في القائمة المنسدلة اليمنى اختر الحدث "دخول الأداة" Enter.. هذا هو الحدث الذي سنغيّر فيه لون مربّع النصّ، فهو ينطلق كلّما دخل مؤشّر الكتابة إلى مربّع النصّ.. اكتب في هذا الحدث الكود التالي:

```
Private Sub FocusedTextBox_Enter(
        ByVal sender As Objec, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Enter
    Me.BackColor = Color.Cyan
End Sub
```

لاحظ أنّ كلمة Me هنا تشير إلى أداة المستخدم، التي هي في حالتنا هذه هي مربّع النصّ.

كذلك يجب برمجة حدث مغادرة مؤشّر الكتابة للأداة Leave، بحيث نعيد فيه لون مربّع النصّ إلى طبيعته.

```
Private Sub FocusedTextBox_Leave(
            ByVal sender As Object, _
            ByVal e As System.EventArgs) _
            Handles MyBase.Leave
    Me.BackColor = Color.White
End Sub
```

كتابة بعض الخصائص:

ما رأيك الآن أن نضيف بعض الخصائص الجديدة لمربّع النصّ؟

سننشئ خاصيّة تسمّى "إلزامي" Mandatory، عند جعل قيمتها True سنغيّر لون مربع النصّ إذا كان فارغا، حتّى نشعر المستخدم بحتميّة إدخال قيمة فيه.. ولكي نسمح للمبرمج باختيار لون هذه الخاصيّة، فسننشئ خاصيّة أخرى تسمّى MandatoryColor.. وبالمرّة سننشئ خاصيّة تسمح بتحديد لون مربّع النصّ عند دخول مؤشّر الكتابة إليه، وسنسمّيها EnterFocusColor، وخاصيّة أخرى تسمح بتحديد لون مربّع النصّ عند خروج مؤشّر الكتابة منه، وسنسمّيها LeaveFocusColor.

تعال نكتب هذه الخصائص:

ابدأ بتعريف المتغيّرات الخاصّة التي ستحتفظ فيها بقيمة كلّ خاصيّة.. لقد جرى العرف البرمجيّ على أن تكون لهذه المتغيّرات نفس أسماء الخصائص ولكن مسبوقة بالعلامة "_":

```
Dim _mandatory As Boolean
Dim _enterFocusColor, _leaveFocusColor As Color
Dim _mandatoryColor As Color
```

أمّا بالنسبة لكود الخصائص، فهو مباشر جدّا ومتماثل، لذلك فسنكتفي بذكر كود خاصيّتين فقط:

```
Property Mandatory ( )As Boolean
    Get
        Mandatory = _mandatory
    End Get
    Set(ByVal Value As Boolean)
        _mandatory = Value
    End Set
End Property
Property EnterFocusColor ( )As Color
    Get
        EnterFocusColor = _enterFocusColor
    End Get
    Set(ByVal Value As Color)
        _enterFocusColor = EnterFocusColor
    End Set
End Property
```

وأخيرا يجب تعديل الحدثين Enter و Leave ليتعاملا مع هذه الخصائص:

```
Private Sub FocusedTextBox_Enter(
        ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Enter
    Me.BackColor = _enterFocusColor
End Sub
```

```
Private Sub FocusedTextBox_Leave( _
        ByVal sender As Object_ ،
        ByVal e As System.EventArgs) _
        Handles MyBase.Leave
    ' إذا كانت خاصية الإلزام صحيحة، وكان مربّع النصّ فارغا أو به مسافات
    ' فغيّر لون مربّع النصّ إلى لون الإلزام
    If _mandatory AndAlso Trim(Me.Text) = "" Then
        Me.BackColor = _mandatoryColor
    Else ' غير لون مربّع النصّ إلى لون المغادرة
        Me.BackColor = _leaveFocusColor
    End If
End Sub
```

والآن لتجربة هذه الإضافات، أعد بناء الأداة من جديد، وانتقل إلى نموذج الاختبار.. ضع عدّة نسخ من أداتنا عليه، وغيّر الخصائص الجديدة التي أنشأناها لكلّ منها باستخدام نافذة الخصائص.. ثمّ شغّل البرنامج واختبر هذه الخصائص.

رائع وسهل وممتع هو تطوير الأدوات.. أليس كذلك؟

تبويب الخصائص:

إذا كنت تعرض الخصائص في نافذة الخصائص مرتّبة على حسب النوع، فستجد أنّ الخصائص التي أنشأناها قد ظهرت كلّها تحت النوع Misc.. ولتغيير هذا الوضع الافتراضيّ، يمكن كتابة بعض السمات attributes أمام تعريف الخاصيّة.. فمثلا لجعل الخاصيّة تظهر تحت مقطع "المظهر" Appearance، استخدم السمة التالية:

```
<Category("Appearance")>
```

ويمكنك أن تضيف خصائصك لنوع جديد غير موجود سابقا في نافذة الخصائص، حيث سيتمّ إنشاؤه من أجلك، وذلك بمجرّد كتابة اسم النوع الجديد كالتالي:

```
<Category("Mine")>
```

ولاستخدام السمات، لا بدّ أن تستورد الفئة System.ComponentModel، فكلّ السمات هي جزء منها:

```
Imports System.ComponentModel
```

وهناك سمة أخرى تهمّك، هي وصف الخاصيّة، ذلك الذي يظهر في الجزء السفليّ من نافذة الخصائص، ليشرح وظيفة الخاصيّة.. انظر الآن كيف سنعدّل تعريف خاصيّة Mandatory، لنضيف هاتين السمتين:

```
<Description("توضّح إذا كان من الممكن ترك الأداة فارغة أم لا"), _
    Category("Appearance")> _
Property Mandatory( ) As Boolean
```

أمّا أهمّ سمة، فهي سمة القيمة الافتراضيّة للخاصيّة DefaultValue، حيث يمكنك كتابة القيمة الافتراضيّة بين القوسين:

```
<Description(" توضّح إذا كان من الممكن ترك الأداة فارغة أم لا"), _
    Category("Appearance"),
    DefaultValue(False)> _
    Property Mandatory( ) As Boolean
```

وهناك سمات لفئة الأداة نفسها.. فمثلا، يمكنك أن تجعل الخاصيّة Mandatory هي الخاصيّة الافتراضيّة، عدّل تعريف الفئة كالتالي:

```
<DefaultProperty("Mandatory")> _
Public Class FocusedTextBox
```

ويمكنك اختبار كلّ ذلك في تطبيق FocusedTextBox في مجلّد برامج هذا الفصل.

حل مشكلة اليمين لليسار في بعض الأدوات:

في الفصل الحادي عشر، سبق أن تكلمنا عن عرض الشجرة من اليمين لليسار، وذلك باستخدام تقنية المرآة عبر دوال API.

لكن لحسن الحظّ أننا نملك طريقة أخرى أكثر أمانا، يمنحها لنا إطار العمل.. فأنت تعرف أن دوال API خاصة بالويندوز وتعتمد على نوع الإصدار.. لهذا فإن الكود الذي يستخدمها هو كود غير مدار Unmanaged Code.. بينما الكود الذي يعتمد على أدوات إطار العمل يسمّى كودا مدارا Managed Code، وهو أكثر أمنا وسهولة من الكود غير المدار.

هنا سنرى كيف نستخدم تقنية المرآة عبر إطار العمل.. المشكلة أنّنا هنا نحتاج لإنشاء أداة جديدة تستخدم هذه التقنية، على عكس ما كنا نفعله بدوال API التي تؤثر على الأدوات الموجودة مباشرة.

تعال نرى كيف نطور أداة الشجرة، لتعرض عناصرها من اليمين لليسار، عندما تتغير قيمة خاصية RightToLeft إلى True.

ابدأ مشروع أداة ويندوز جديدا، وسمّه RightToLeftTree (ستجد هذا المشروع في مجلد برامج هذا الفصل).

افتح نافذة محرّر الكود للأداة.. وغير السطر:

Inherits System.Windows.Forms.UseControl

إلى:

Inherits System.Windows.Forms.TreeView

الآن أداتنا تمثّل الشجرة التقليدية.. نحتاج لإضافة الكود الذي يصحّح عمل خاصية RightToLeft.

هذا هو الكود:

```
Const WS_EX_LAYOUTRTL As Integer = &H400000
Const WS_EX_NOINHERITLAYOUT As Integer =
        &H100000

Protected Overrides ReadOnly Property CreateParams( ) _
      As System.Windows.Forms.CreateParams
        Get
            Dim MirrorExStyle As CreateParams
            CreateParams = MyBase.CreateParams
            If Me.RightToLeft Then
                MirrorExStyle.ExStyle =
                MirrorExStyle.ExStyle Or _
                    WS_EX_LAYOUTRTL Or _
                    WS_EX_NOINHERITLAYOUT
            End If
          Return MirrorExStyle
       End Get
End Property
```

أظنّ الكود واضحا.. لقد قمنا باستبدال Override خاصية إنشاء المعاملات CreateParams.. هذه الخاصية خاصة بالنماذج والأدوات.. وهي تُستدعى عند تغيّر خصائص الأداة وعند رسمها.. كل ما سنفعله، هو أن نحصل على معاملات الأداة عن طريق الخاصية الأصلية

```
CreateParams = MyBase.CreateParams
```

حيث MyBase يشير إلى الفئة الأم التي نرث منها وهي هنا فئة الشجرة.. بعد هذا نتأكد من قيمة خاصية RightToLeft للشجرة.. فإن كانت True، فعلينا أن نعرض الشجرة من اليمين لليسار، وذلك بإجراء عملية Or بين قيمة المعاملات الحالية للشجرة وبين الثابت WS_EX_LAYOUTRTL الذي يدلّ على عرض الشجرة من اليمين

لليسار (لماذا لا نستخدم عملية تساوي مباشرة؟.. حتّى نحافظ على باقي قيم المعاملات بدون تغيير).

أمّا الثابت WS_EX_NOINHERITLAYOUT فهو يمنع توريث هذا المظهر للأدوات المحتواة داخل هذه الأداة إن وجدت.. (النموذج واللوحة Panel أدوات حاوية لغيرها Container).. طبعا يمكن التخلص من هذا الثابت، فهو لا يفيدنا مع الشجرة.. ولكنه يفيدنا مع النموذج (لو أردت تعريبه لتظهر أيقونة التحكم على اليمين، وأزرار التكبير والتصغير على اليسار) دون أن تتأثر بذلك الأدوات الموجودة عليه.

هل معنى هذا أن هذه الطريقة صالحة لاستخدامها مع أيّ أداة أخرى؟

هذا صحيح.. وبنفس الكود بدون تغيير أي حرف، سوى اسم الأداة فقط!.. بهذا يمكنك تطوير العديد من الأدوات التي لا يتم عرضها من اليمين لليسار، مثل ProgressBar و ToolBar وغيرها.

الأمر بسيط كما ترى.

بقي شيء صغير.. عند عكس عرض الشجرة، سيفسد عرضها في صورة مجسمة (نتيجة عكس موضع الخط الأبيض مع الخط الأسود الذي يوحي بالعمق).. لهذا فمن الأفضل ألا تعرض هذه الأداة مجسمة.. لهذا يجب أن تضيف هذا السطر للحدث New الخاص بأداتنا المطوّرة:

Me.BorderStyle = BorderStyle.FixedSingle

ولو أردت، يمكنك استخدام وسائل الرسم والتلوين لرسم خط أسود على الحافة اليسرى للشجرة، وآخر على الحافة العليا، وخطين رماديين على الحافتين اليمنى والسفلى، ليوحي هذا بالعمق، فتبدو الشجرة مجسمة.

جرّب الآن أن تضيف مشروع اختبار للأداة، وضع نسخة منها على النموذج، وأضف إليها عدة عناصر (من خاصية Nodes)، ثم غيّر قيمة خاصية RightToLeft إلى True ثمّ إلى False ثمّ إلى True، وانظر كيف سيتغير عرض الشجرة أثناء التصميم.. ثمّ شغّل التطبيق (بعد جعل تطبيق الاختيار هو تطبيق البداية) وانظر لأداتنا الرائعة.

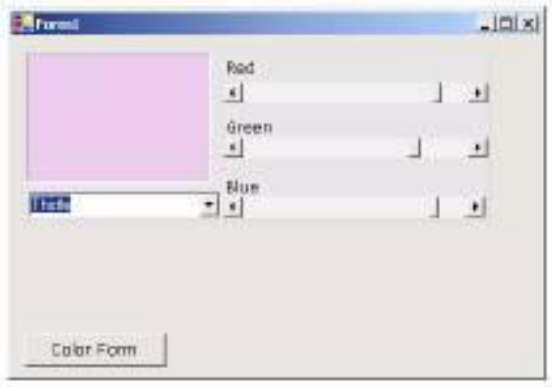
إنشاء أدوات مركّبة

في هذا المقطع لن نرث أداةً ونطوّرها، ولكنّنا سنستخدم مجموعة من الأدوات تقوم بوظيفة معيّنة، لإنشاء أداة جديدة مركّبة.

تعال نرى كيف ننشئ أداة تسمح للمستخدم باختيار اللون.. هذه الأداة ستتكوّن من الأجزاء التالية:

- ثلاثة منزلقات أفقيّة، تمثّل نسب الألوان الأساسيّة: الأحمر والأخضر والأزرق، بحيث يستطيع المستخدم تكوين اللون من نسبه.
- قائمة مركّبة منسدلة، يختار المستخدم منها اسم اللون مباشرة.
- لافتة سنجعل لون خلفيّتها هو اللون الذي سيركّبه المستخدم أو يختاره، حتّى يستطيع معاينة اللون.

افتح مشروع أداة ويندوز جديدا وسمّه ColorEdit، وصمّم أداة المستخدم لتبدو كما في الصورة.



سمّ المنزلقات الأفقيّة: RedBar، و GreenBar و BlueBar، واجعل قيمة خاصيّة MinimumValue لكلّ منها صفرا، وقيمة خاصيّة

MaximumValue لكـــلّ منهـــا ٢٥٥.. وســـمّ القائمـــة المركّبـــة NamedColors.. ولملء هذه القائمة بالألوان، استخدم الإجراء التالي:

```
Private Sub AddNamedColors( )
    With ComboBox1.Items
        .Add("Red")
        .Add("Yello")
        .Add("Blue")
        ' أضف ما تريد من أسماء الألوان بنفس الطريقة
        ' واجعل آخر عنصر في القائمة هو "غير معروف"
        ' لأنّ هناك ملايين الألوان التي لا تمتلك أسماء!
        .Add("غير معروف")
    End With
End Sub
```

ويمكنك الاسترشاد بأسماء الألوان التي يحتوى عليهـــا الســـجلّ Color، وستجدها في ملفّات إرشادات اللغة.

ملحوظة:

لو أردت أن تضيف أسماء الألوان التي يحتوى عليها السجلّ Color ــ وهي بالإنجليزيّة ــ للقائمة، فعليك أن تستخدم الكود التالي:

```
Dim CNames( ) As String =
    [Enum].GetNames(GetType(KnownColor))
ComboBox1.Items.AddRange(CNames)
ComboBox1.Items.Add("غير معروف")
```

مع ملاحظة أنّ المرقّم KnownColor لا يمدّك فقـــط بأســـماء ١٢٨ لونا، بل يمدّك كذلك بأسماء ألوان الويندوز التـــي يســـمح للمســـتخدم باختيارها من خصائص سطح المكتب DeskTop Properties، مثـــل

ActiveBorder و ControlLight... إلخ.. وهي تقع في بداية المرقّم KnownColor (من ٠ إلى ٢٦).

وهذا يفيد في ترك الحرّية للمستخدم في تخصيص ألوان البرنامج كجزء من الويندوز ككلّ.

ولكي يتمّ تكوين الألوان في هذا الحالة، يجب استخدام الكود التالي:

```
Clr = Color.FromKnownColor(ComboBox1.SelectedIndex)
```

مع ملاحظة حتميّة استخدام معالجات الاستثناء اللازمة، لتوقّي أيّ خطأ، مثل عدم قابلية بعض الأدوات للون الشفاف.

وللتعامل مع أداتنا، سنجعل لها خاصيتي "اللون المحدّد" SelectedColor و"اللون المسمّى" NamedColor، ها هو ذا كود الخاصيّة الأولى:

```
Property SelectedColor( ) As Color
    Get
        ' تعيد هذه الخاصيّة لون اللافتة
        SelectedColor = Label1.BackColor
    End Get

    Set(ByVal Value As Color)
        ' عند تغيير قيمة هذه الخاصيّة يجب أن تظهر مكوّنات
        اللون الجديد على المنزلقات
        RedBar.Value = Value.R
        GreenBar.Value = Value.G
        BlueBar.Value = Value.B
    End Set
End Property
```

أما خاصيّة NamedColor فسنجعلها للقراءة فقط، وكودها كالتالي:

```
ReadOnly Property NamedColor( ) As String
    Get
        ' هذه الخاصيّة ترجع قيمة العنصر المحدّد حاليّا في القائمة
        NamedColor = ComboBox1.SelectedItem
    End Get
End Property
```

وعندما يختار المستخدم اسم لون من القائمة، فعليك أن تكوّن اللون المقابل للاسم، باستخدام الوسيلة Color.FromName، إلا في حالة اختيار العنصر "غير معروف"!

```
Private Sub ComboBox1_SelectedIndexChanged(
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles _
        ComboBox1.SelectedIndexChanged
    Dim namedColor As Color
    Dim colorName As String = ComboBox1.SelectedItem
    If colorName <> "غير معروف" Then
        ' لن يفيد هذا الكود لو كتبت الأسماء باللغة العربيّة
        ' أترك لخيالك حلّ هذه المشكلة
        namedColor =
        Color.FromName(colorName)
        RedBar.Value = namedColor.R
        GreenBar.Value = namedColor.G
        BlueBar.Value = namedColor.B
    End If
End Sub
```

بقيت آخر خطوة، وهي كتابة الكود المستجيب لانزلاق المنزلقات الثلاثة:

```
Private Sub ColorScroll(ByVal sender As Object,
            ByVal e As ScrollEventArgs) _
            Handles RedBar.Scroll, GreenBar.Scroll,
            BlueBar.Scroll
        Label1.BackColor = Color.FromARGB(
            HScrollBar1.Value, _
            HScrollBar2.Value, HScrollBar3.Value)
End Sub
```

الآن لم يبقَ إلا اختبار أداة الألوان.. ولفعل ذلك، أضف مشروع اختبار للتطبيق (مشروع ويندوز عادي، سنستخدمه للاختبار)، واجعله مشروع بدء التشغيل.. قم ببناء الأداة وأضف نسخة منها لنموذج الاختبار.. شغّل التطبيق وانظر كيف يمكنك اختيار الألوان باستخدام المنزلقات أو القائمة.

أنهِ التطبيق وأضف زرّا للنموذج، واكتب فيه الكود التالي، لتغيير لون خلفيّة النموذج تبعا للون الأداة:

```
Private Sub Button1_Click(ByVal sender As Object, _
            ByVal e As System.EventArgs) _
            Handles Button1.Click
        Me.BackColor = ColorEdit1.SelectedColor
End Sub
```

بناء أدوات بأشكال خاصّة

في هذه المرّة لن نضع أيّ أداة على أداة المستخدم، بل سنقوم نحن بالرسم عليها، عن طريق إعادة كتابة وسيلة الرسم OnPaint لتستبدل Overrides وسيلة الرسم الخاصّة بالأداة.. تعال نرى ذلك عمليّا:

اللافتة المجسّمة

سنقوم هنا بإنشاء لافتة تعرض نصّا مجسّما.. وأنصحك هنا بمراجعة فصل التلوين، فقد قمنا فيه بإنشاء مشروع Text Effects الذي يرسم نصّا مجسّما على النموذج.. سنقوم هنا بنفس الأمر، لكنّنا سنرسم في هذه المرّة على أداة المستخدم وليس على نموذج.

ابدأ مشروع أداة ويندوز جديدا وسمّه Label3D.. سمِّ أداة المستخدم أيضا Label3D.

هيّا بنا نرى كيف نصمّم هذه الأداة الشيّقة:

إضافة الخصائص:

إنّ من الخصائص التي سنضيفها لهذه اللافتة، خاصيّة المحاذاة، وهي تأخذ عدّة قيم، سنضعها في المرقّم Align.. ابدأ بتعريف هذا المرقّم في بداية فئة الأداة:

```
Public Enum Align
    TopLeft
    TopMiddle
    TopRight
    CenterLeft
    CenterMiddle
    CenterRight
    BottomLeft
    BottomMiddle
    BottomRight
End Enum
```

أمّا النصّ الذي ستعرضه اللافتة، فسنسمح للمبرمج بتحديد التأثير الذي سيعرض به، وذلك من خلال خاصيّة Effect3D التي تأخذ إحدى قيم المرقّم التالي:

```
Public Enum Effect3D
    None ' بدون تأثير
    Raised ' مرتفع
    RaisedHeavy ' مرتفع سميك
    Carved ' محفور
    CarvedHeavy ' محفور سميك
End Enum
```

بعد ذلك سنعرّف متغيّرين خاصّين، نحفظ فيهما قيمتي الخاصيتين.. لا بدّ بالطبع أنّ يكون كل متغيّر من نوع المرقّم الذي سيحفظ قيمته:

```
Private Shared mAlignment As Align
Private Shared mEffect As Effect3D
```

الآن سنكتب خاصيتي المحاذاة والتأثير المجسّم:

```
Public Property Alignment( ) As Align
    Get
        Alignment = mAlignment
    End Get
    Set(ByVal Value As Align)
        mAlignment = Value
        Invalidate( )
    End Set
End Property

Public Property Effect( ) As Effect3D
    Get
        Effect = mEffect
    End Get
    Set(ByVal Value As Effect3D)
        mEffect = Value
        Invalidate( )
    End Set
End Property
```

لاحظ استخدامنا للوسيلة Invalidate، حتّى تعيد رسم النصّ من جديد على سطح اللافتة، وبهذا يتمّ تطبيق القيمة الجديدة للخاصيّة على النصّ.

أضف مشروع اختبار للتطبيق، وحاول أن تجرّب الأداة.. ضع نسخة منها على النموذج، وانظر لخصائصها في نافذة الخصائص.. ستكتشف أنّ النافذة لا تحتوي فقط على الخاصيتين اللتين عرّفناهما، بل إنّ هناك عددا هائلا من الخصائص الجاهزة التي تتعامل مع شكل وموقع اللافتة وألوانها

وخطوطها.. هذه الخصائص لم تكتبها أنت، ولكنّ أداة المستخدم منحتهـا لك لحسن الحظّ.

ولكن رغم هذا ما زلنا بحاجة لخاصـيّة Text3D التـي سـيكتب فيهـا المستخدم النصّ الذي يريد عرضه في اللافتة.. تعال نكتب هذه الخاصيّة:

```
Private mText As String
Public Property Text3D ( ) As String
        Get
                Text3D = mText
        End Get

        Set(ByVal Value As String)
                mText = Value
                Invalidate( )
        End Set
End Property
```

رسم النصّ المجسّم:

الآن حان وقت الجدّ.. نحتاج لكتابة الوسيلة OnPaint.. هــذه الوســيلة ستستدعيها الأداة تلقائيّا كلّما احتاجت لأن يعاد رســمها وعنــد اســتخدام الوسيلة Invalidate.. ابدأ بوضع الجملة التالية في بداية الملفّ:

```
Imports System.Drawing.Drawing2D
```

والآن تعال نرى كيف سنرسم النصّ بتأثيراته ومحاذاته:

```
Protected Overrides Sub OnPaint(ByVal e As _
        PaintEventArgs)
    Dim lblFont As Font = Me.Font
    Dim lblBrush As New SolidBrush(Color.Red)
    Dim X, Y As Integer
    ' يجب معرفة أبعاد النصّ حتّى نستطيع محاذاته
    Dim TextSize As SizeF
    SizeF = e.Graphics.MeasureString(mText, lblFont)
    Select Case Me.mAlignment
        Case Align.BottomLeft
            ' رسم النصّ في الركن السفليّ الأيسر
            X = 2 ' نقطتان لمراعاة حافة الأداة
            Y = Me.Height - TextSize.Height
        Case Align.BottomMiddle
            ' رسم النصّ في أسفل المنتصف
            X = CInt((Me.Width - TextSize.Width) / 2)
            Y = Me.Height - TextSize.Height
        Case Align.BottomRight
            ' رسم النصّ في الركن السفليّ الأيمن
            X = Me.Width - TextSize.Width - 2
            Y = Me.Height - TextSize.Height
```

```
        Case Align.CenterLeft
            ' رسم النصّ في منتصف اليسار
            X = 2
            Y = (Me.Height - TextSize.Height) / 2
        Case Align.CenterMiddle
            ' رسم النصّ في مركز اللافتة
            X = (Me.Width - TextSize.Width) / 2
            Y = (Me.Height - TextSize.Height) / 2
        Case Align.CenterRight
            ' رسم النصّ في منتصف اليمين
            X = Me.Width - TextSize.Width - 2
            Y = (Me.Height - TextSize.Height) / 2
        Case Align.TopLeft ' رسم النصّ في أعلى اليسار
            X = 2
            Y = 2
        Case Align.TopMiddle
            ' رسم النصّ في أعلى المنتصف
            X = (Me.Width - TextSize.Width) / 2
            Y = 2
        Case Align.TopRight ' رسم النصّ في أعلى اليمين
            X = Me.Width - TextSize.Width - 2
            Y = 2
    End Select
    ' المتغيّران التاليان سنضع فيهما
    ' مقدار إزاحة النصّ رأسيّا أو أفقيّا حتّى يبدو مجسّما
    Dim dispX, dispY As Integer
```

```
        Select Case mEffect
            Case Effect3D.None
                dispX = 0 : dispY = 0
            Case Effect3D.Raised
                dispX = 1 : dispY = 1
            Case Effect3D.RaisedHeavy
                dispX = 2 : dispY = 2
            Case Effect3D.Carved
                dispX = -1 : dispY = -1
            Case Effect3D.CarvedHeavy
                dispX = -2 : dispY = -2
        End Select
        ' رسم النصّ بلون أبيض
        lblBrush.Color = Color.White
        e.Graphics.DrawString(mText, lblFont,
            lblBrush, X, Y)
        ' رسم النصّ مرّة أخرى بلون أسود، 
        ' مع إزاحته قليلا رأسيا وأفقيّا ليبدو التأثير المجسّم
        lblBrush.Color = Me.ForeColor
        e.Graphics.DrawString(mText, lblFont,
            lblBrush, X + dispX, Y + dispY)
    End Sub
```

الآن حاول تجربة ما فعلناه.. ستكتشف كم هو رائع!

حاول كذلك أن ترى أحداث اللافتة المجسّمة.. ستكتشف أنّ لها عددا هائلا من الأحداث الجاهزة التي لم تكتبها أنت.. منتهى الراحة!

جعل خلفيّة الأداة شفّافة:

ما رأيكُ أن نُضيفَ إمكانيّة جديدةً للافتة المجسّمة، بحيثُ يمكنك أن تجعل خلفيّتها شفّافة؟.. إنّ ذلك سيضمنُ مظهرا أفضلَ للنموذج عند وضع صورة في خلفيّته، حيثُ لن تحجب اللافتة ذلك الجزء من الصورة الذي يقع وراءها.

عامّةً الأمرُ في غاية البساطة: اكتب الجملة التالية في الحدث New للأداة:

Setstyle(ControlStyles.SupportsTransparentBackColor, True)

إنّ هذه الجملة ستسمح لك بجعل خلفيّة النموذج أو الأداة شفّافة.. ويمكنك أن تجرّب ذلك.. قم ببناء الأداة من جديد، ثمّ انتقل لنموذج الاختبار.. حدّد اللافتة المجسّمة، وفي نافذة الخصائص حدّد خاصيّة "لون الخلفيّة" BackColor، وفي خانة القيمة اضغط زرّ الإسدال.. في مربّع الألوان اضغط شريط Web.. ستجد أنّ أول قيمة في هذا الشريط هي "شفّاف" Transparent.. اختر هذه القيمة.. وحتّى تتأكّد بالفعلِ أنّ خلفيّة اللافتة شفّافة، غيّر لون النموذج، أو أضف لخلفيّته صورة (عن طريق الخاصيّة BackgroundImage).

رأيت؟.. غاية البساطة!

وضع القيم المبدئيّة للأداة:

لتحديد القيم المبدئيّة للأداة، تلك التي ستأخذها خصائصها عند وضعها على النموذج لأوّل مرّة، استخدم الحدث New.. وستجد اسمه في القائمة المنسدلة اليمنى في محرّر الكود.. اضغط هذا الاسم للوصول إليه، واكتب فيه ما يلي:

```
Public Sub New( )
    ' ********** لا تغيّر المقطع التالي ولا تحذفه ******
    MyBase.New( )
    ' This call is required by the Windows Form Designer.
    InitializeComponent( )
    '*************************************
    mText = "Label3D"
    mAlignment = Align.CenterMiddle
    mEffect = Effect3D.Raised
    ' لاحظ أنّ بإمكانك تحديد الخطّ الافتراضي من نافذة خصائص أداة المستخدم
    Me.Font = New Font(Me.Font.Name, 14,
                FontStyle.Bold)
    ' الجملة التالية تجبر الأداة على استدعاء حدث الرسم عندما يتغيّر حجمها
    SetStyle(ControlStyles.ResizeRedraw, "True")
End Sub
```

ومن الجدير ذكرا، هو أنّ أداة المستخدم User Control تمتلك خاصيّة جاهزة تريحك من استدعاء الوسيلة SetStyle.. هذه الخاصيّة هي ResizeRedraw، والتي عند جعل قيمتها True تجبر الأداة على استدعاء حدث الرسم عندما يتغيّر حجمها.. ويمكنك استبدال الجملة التالية بجملة SetStyle في كود الإجراء New:

```
Me.ResizeRedraw = True
```

أحداث تغيّر الخصائص XChanged:

ستجد أنّ الأداة تمنحك العديد من الأحداث الجاهزة، معظمها يحدث عند تغيّر خاصيّة من الخصائص، مثل تغيّر الخط FontChanged.. على نفس هذا المنوال لا بدّ أن تكتب أحداث تغيّر الخصائص الجديدة التي كتبتها للأداة.

ابدأ بتعريف الأحداث كالتالي:

```
Public Event AlignmentChanged(ByVal sender As _
    Object, ByVal e As EventArgs)
Public Event EffectChanged(ByVal sender As Object, _
    ByVal e As EventArgs)
Public Event TextChanged(ByVal sender As Object, _
    ByVal e As EventArgs)
```

ثم أطلق كل حدث منها في مقطع تغيّر الخاصيّة المناظرة له.. اكتب السطر المناسب من السطور الثلاثة التالية بعد السطر الذي يستدعي الوسيلة Invalidate في الخاصيّة المناظرة لكلّ حدث:

```
RaiseEvent AlignmentChanged(Me, _
    New EventArgs( ))
RaiseEvent EffectChanged(Me, New EventArgs( ))
RaiseEvent Text3DChanged(Me, New EventArgs( ))
```

خصائص معامل الحدث:

أعرف أنّك ستتساءل عن جدوى إرسال المعامل الثاني للحدث ما دام لا يحتوي على أي معلومات مفيدة؟

عامّة يمكنك تغيير ذلك.. فلو أردت إرسال أيّ معلومة جديدة في المعامل الثاني، فعليك أن تنشئ فئة جديدة ترث الفئة EventArgs، حيث تضيف

لها كلّ ما تريد من الخصائص، لتحمل المعلومات التي تريـــد تمريرهـــا للمبرمج عند انطلاق الحدث.. افترض مثلا أنّك تريد أن تخبر المبـــرمج بالقيمة القديمة والقيمة الجديدة للخاصيّة، مع منحه القدرة على إلغاء التغيير قبل حدوثه.. في هذه الحالة عرّف الفئة التالية:

```
Public Class PropertyChangedEvent
    Inherits EventArgs
    Public Shared ReadOnly OldValue As Object
    Public Shared ReadOnly NewValue As Object
    Public Shared Cancel As Boolean
End Class
```

الآن يجب أن نعدّل تعريف الأحداث ليصبح المعامل الثاني مـــن النـــوع PropertyChangedEvent.. تعال نأخذ الحدث AlignmentChanged كمثال، وعليك فعل المثل مع الحدثين الآخرين:

```
Public Event AlignmentChanged(ByVal sender As _
    Object, ByRef e As PropertyChangedEvent)
```

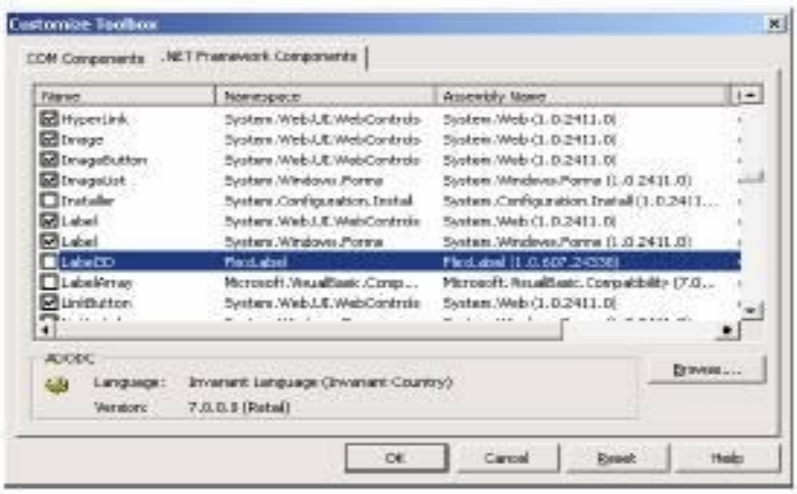
ويجب إعادة كتابة خاصيّة المحاذاة لتصير كالتالي:

```
Public Property Alignment( ) As Align
    Get
        Alignment = mAlignment
    End Get
    Set(ByVal Value As Align)
        Dim e As New PropertyChangedEvent
        e.OldValue = mAlignment
        e.NewValue = Value
        RaiseEvent AlignmentChanged(Me, e)
        If e.Cancel = False Then
            mAlignment = Value
            Invalidate( )
        End If
    End Set
End Property
```

استخدام اللافتة المجسمة في مشاريع أخرى:

لكي تستخدم اللافتة المجسّمة في مشروع آخر اتبع هذه الخطوات:

- ابدأ مشروعا جديدا، واعرض نموذجه.
- اضغط صندوق الأدوات بزر الفأرة الأيمن، ومن القائمة الموضعيّة اضغط الأمر Customize Toolbox.



- في مربّع الحوار الذي سيظهر لك، اضغط الشريط ..NET Framework Components
- اضغط زرّ التصفّح Browse، وفي مربّع حوار "فتح ملف" افتح الملفّ Label3D.dll (ستجده في المجلّد Bin في مجلّد مشروع اللافتة المجسمة).
- ستجد أنّ اللافتة المجسّمة قد أضيفت لقائمة .NET Framework components.. تأكّد أن مربّع الاختيار أمام اسمها عليه علامة (**ü**)، واضغط OK.
- الآن ستظهر أيقونة اللافتة المجسّمة في صندوق الأدوات، حيث يمكنك استخدامها بالطريقة التقليديّة.

تصميم أدوات بأشكال غير تقليديّة:

رأيت في الفصل الخامس كيف أنشأنا نموذجا بيضاويّا.. لا يوجد ما يمنع من تنفيذ نفس الأمر مع أداة المستخدم لتبدو بيضاويّة أو تتخذ أيّ شكل تريده.

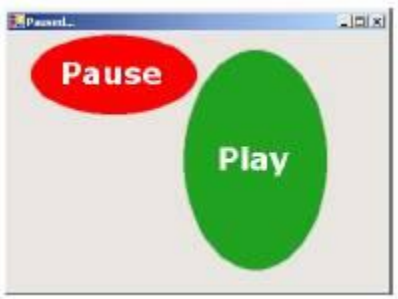
إنّنا سنقوم هنا بالمثل، ولكن ليس باستخدام خاصيّة TransparencyKey، ولكن عن طريق استخدام خاصيّة "منطقة الرسم" Region الخاصّة بأداة المستخدم، والتي عن طريقها نستطيع تحديد منطقة الأداة.

ملحوظة:

يمتلك النموذج أيضا خاصيّة Region، حيث يمكنك استخدامها لتغيير شكل النموذج، بنفس الطريقة التي سنشرحها مع أداة المستخدم هنا.. ولكنّ هذه الخاصيّة لا توجد في الوضع التلقائيّ في قائمة أعضاء النموذج (التي تظهر أثناء كتابة الكود).. ويمكنك أن تعرض كلّ خصائص ووسائل النموذج في قائمة الأعضاء، باتّباع التالي: افتح القائمة Tools واضغط الأمر Options.. في النافذة التي ستظهر لك، اضغط العنصر Text Editor، ومن عناصره الفرعيّة اختر Basic ومن عناصره الفرعيّة اختر General.. على اليسار ستجد مربّعات اختيار.. أزل العلامة من الاختيار التالي "إخفاء الأعضاء المتقدّمة" Hide Advanced Members.. إنّ هذا سيؤدّي لظهور كلّ أعضاء النموذج والأدوات في قائمة الأعضاء، ومن بينها خاصيّة Region.

انظر للمثال التالي:

```
Protected Overrides Sub OnPaint(
            ByVal e As PaintEventArgs)
        Dim G As Graphics =  Me.CreateGraphics
        Dim roundPath As New GraphicsPath( )
        Dim R As New Rectangle(0, 0, Me.Width,
        Me.Height)
        roundPath.AddEllipse(R)
        Me.Region = New Region(roundPath)
        ' الخطوة التالية اختياريّة، وهي رسم حافة رمادية حول منطقة الأداة
        Me.CreateGraphics.DrawEllipse(New
        Pen(Color.DarkGray, 3), R)
End Sub
```



وللتدريب، لديك في مجلّد برامج هذا الفصل تطبيــق RoundButton.. وقتا ممتعا.

الأدوات من النوع ActiveX:

لعلّك لاحظت أنّ الأدوات الجديدة التي تنشئها بـ VB.Net لها الامتداد dll. .. لم يكن هذا هو الحال في النسخ السابقة، حيث كانت التقنية المسمّاة COM هي المستخدمة، وكانت الأدوات تسمّى ActiveX ولها الامتداد .ocx. .

ولكن.. هل معنى هذا أن تقنية COM قد أقصيت نهائيّا؟

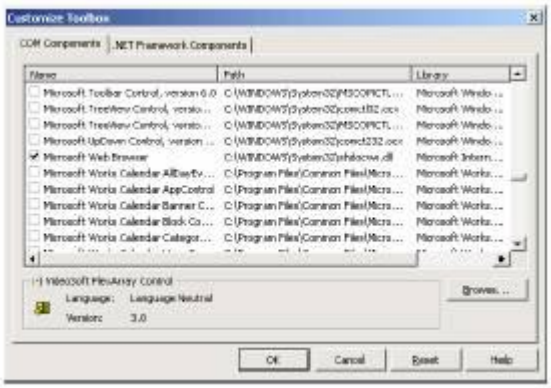
لا.. إنّ هذا مستحيل، نظرا لأنّ هناك ملايين الأدوات التي تمّ إنتاجها من تلك النوعيّة، ممّا يعني أنّ إهمالها مرّة واحدةً سيكون خسارة فادحةً للمبرمجين.

عامّة، كانت (ميكروسوفت) تطلق على تقنية Net. في بدايتها COM2، حيث قامت بإعادة بناء تقنية COM من جديد، لحلّ المشاكل التي كانت تلازمها.

الخلاصة: ما زال باستطاعتك استخدام أدوات COM في VB.Net.

فمثلا يمكنك استخدام أداة تصفّح الإنترنت WebBrowser Control، التي تعتبر بمثابة متصفّح الإنترنت Internet Explorer موضوعا في أداة، وبواسطتها يمكنك فتح صفحات HTML (ويمكنك استغلال ذلك في عرض ملفّات المساعدة)، كما يمكنك الاتصال بمواقع الإنترنت.

ولإضافة هذه الأداة لتطبيقك، اضغط صندوق الأدوات بزر الفأرة الأيمن، ومن القائمة الموضعيّة اضغط Customize Toolbox.. وفي مربّع الحوار اضغط شريط COM Components.. الآن يمكنك أن تضيف الأداة التي ترغب فيها.. ابحث عن الأداة Microsoft Web Browser، فإذا لم تكن موجودة، فاضغط زر Browse، وافتح مجلّد System32 في مجلّد الويندوز، ومنه اختر الملفّ shdocvw.dll.. هذا الملفّ سيكون موجودا بالتأكيد لو كانت لغة VB6 موجودةً على جهازك، فإذا لم تكن موجودة، فلا أضمن إن كنت ستجد هذا الملفّ أم لا.. فإذا وجدته فاختره واضغط موافق.



ستجد اسم الأداة قد أضيف للقائمة وبجوارها علامة (**ü**).. اضغط زر Ok لإغلاق مربّع الحوار.. الآن ستجد أيقونة الأداة Explorer مضافةً كآخر عنصر في صندوق الأدوات.. اضغطها مرّتين لتضيف منها نسخةً للنموذج.. ستلاحظ أنّ اللغة ستستهلك بعض الوقت قبلَ أن تضع هذه النسخة على النموذج.. هذا الوقت تحتاجه اللغة لتكوين بعض الملفّات الوسيطة التي تستطيع من خلالها أن تتحاور مع الكائنات التي تستخدم تقنية Com.

ولفتح ملفّ HTML باستخدام هذه الأداة، استخدم الوسيلة التالية:

AxWebBrowser1.Navigate2(FileName)

حيث FileName هو اسم صفحة HTML التي تريد فتحها.

ولقد رأيت في فصل الرسم والتلوين، كيف استخدمنا الأداة Script control لرسم أيّ دالّة يكتبها المستخدم.

إنّ عالم أدوات ActiveX رحيب، ويمتلئ بالإمكانيّات الرائعة.. وإن كان استخدامك لها يستلزم أن تحصل على ملفّات الإرشادات التي تشرح كيفيّة استخدامها، فهذه المعلومات لن تجدها بالطبع في إرشادات VB.

آخر ما أحبّ أن ألفت نظرك إليه، هو أنّ VB.Net يسمح لك بتحويل الفئات والأدوات التي تنشئها إلى تقنية COM، لتتوافق مع اللغات التي تتعامل مع هذه التقنية.. للأسف.. هذا موضوع معقّد ويحتاج لمرجع أكثر تقدّما.

الفصل الثالث

مشروع تعليميّ لـ Direct3D

مقدمة حول Microsoft DirectX 9.0:

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

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

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

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

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

إنَ حوالي ٧٥% على الأقّل من الألعاب تحتاج لإصدار مـــن إصـــدارات DirectX المختلفة لكي تعمل.. إنّ هذا يشي بوضوح بأهمّيّة هذه التقنية، ومدى الإمكانيّات التي تقدّمها.

مفاهيم أساسيّة

عالم المجسّمات:

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

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

طبعا ستقول في نفسك مستخفّا: وماذا في عدّة دقائق؟

ولكي تعرف الإجابة، يجب أن أذكّرك أنّ الرسوم المتحرّكة عبارة عن مجموعة هائلة من الصور، يتمّ عرضها متتابعة بسرعة (أكثر من ٢٠ لقطة في الثانية)، بحيث تنخدع عين الرائي فترى الرسوم تتحرّك.. هل تخيّلت إذن كم صورة في دقيقة واحدة؟.. أكثر من ١٢٠٠ صورة.. حاول أن تتخيّل إذن عدد الساعات الذي ستستهلكه عمليّة تكوين مشهد رسوم مجسّمة مدّته دقيقة واحدة!

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

ثمّ إنّنا ــ كمبرمجين ــ لن نقوم بهذه العمليّة.. إنّ كلّ ما سنفعله هو استلام المجسّمات من مصمّميها (بالاستعانة بتطبيقات مثل 3DS Max

و TrueSpace و Lightwave و Maya وغيرها)، واستخدامها في تطبيقاتنا كما يحلو لنا.

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

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

ملحوظة١:

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

ملحوظة٢:

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

نظام الإحداثيات ثلاثي الأبعاد 3-D Coordinate Systems:

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

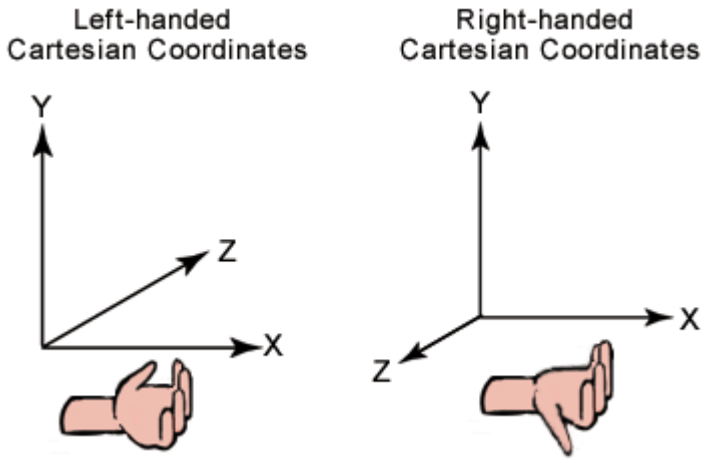
ملحوظة:

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

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

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

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



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

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

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

هذا، ويستخدم Direct3D نظام الإحداثيّات اليساريّ Left-handed coordinate system.

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

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

الرءوس Vertices:

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

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

ولكن لماذا المثلّث بالذات؟

أولا: لأنّه شكل مغلق.

ثانيا: لأنّه يتكوّن من أقلّ عدد من الرءوس.

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

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

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

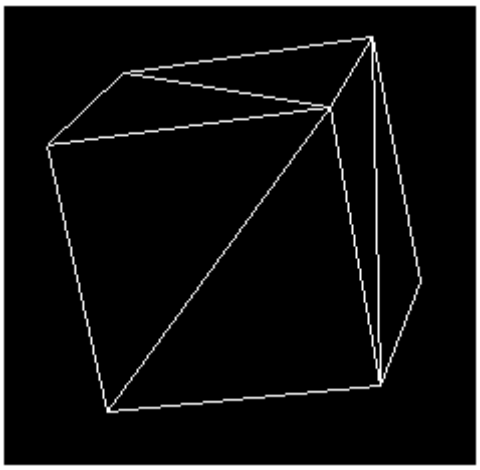
للشكل تبرزه من هذه الزاوية.. وبهذا ستشعر أنت أنّك في عـــالم مجسّـــم حقيقيّ، لأنّ كلّ التفاصيل التي تريدها موجودة!

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

طبعا أنت تعرف أنّ للمثلّث ثلاثة رءوس 3 vertices.

إنّ الرأس vertex هو أبسط وأصغر وحدة سنســتخدمها للتعبيــر عــن البيانات في الرسوم المجسّدة.. هذا الرأس ما هو إلا نقطة في النظام ثلاثيّ الأبعاد، ممّا يعنـــي أنّ تعريفهـــا يحتـــاج لمعرفـــة إحـــداثيّاتها الثلاثـــة: (س، ص، ع).. ولكنّ الإحداثيّات ليست هي كلّ المعلومات التي تخـــصّ النقطة.. هناك أيضا لونها، درجة إضاءتها، خامتها.. إلخ.. وكلّها عوامل تؤثّر في سلوك هذه النقطة عند تحرّكها وتفاعلها مع ما حولها، كما سنرى فيما بعد.

تعال نرى مثالا لكيفيّة تمثيل أحد المجسّمات، وليكن المكعّب:



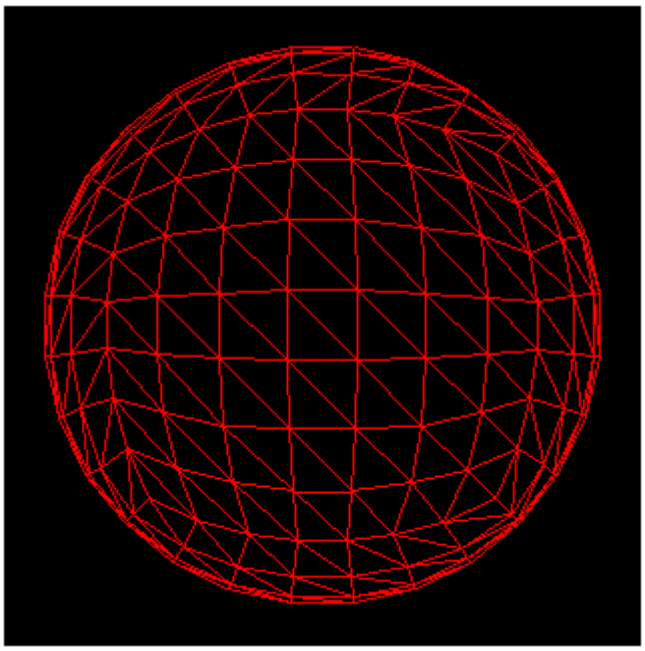
إنّ للمكعّب ستّةَ أوجه، كلّ منها عبارة عن مربّع.. وأنت تعرف أنّ قطر المربّع يقسمه لمثلّثين متماثلين.. معنى هذا أنّ المكعّب يتكوّن من ١٢ مثلّثا.

هل يعني هذا أنّنا نحتاج لـ ٣ × ١٢ = ٣٦ رأسا لتمثيل المكعّب؟

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

إنّنا إذن نحتاج لـ ٨ رءوس فقط، حيث يمكن الحصول على كلّ المثلثات بتوصيل كلّ ثلاثة من هذه الرءوس معا.

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



الخامات Textures:

لن يكون عالمك المجسّم قريبا من الحقيقة، ما لم تراعِ نوعيّة الخامات الداخلة في تكوين عناصره.. وإلا فكيف يمكنك مثلا رسم فقّاعة صابون تعلو في الفضاء.. إنّ هذه الفقّاعة ستعكس بعض المشاهد، وستشفّ عن بعض المشاهد في خلفيّتها.. إنّ من العبث أن تعتقد أنّك ستبرمج كلّ ذلك! كلّ ما عليك فعله، هو أن تستعين بأحد محترفي الرسوم المجسّمة ليعطيك صورة مجسّمة للفقاعة والمكان من حولها.. ثمّ باستخدام Direct3D يمكنك تحديد خامات كلّ مادة ودرجة شفافيتها ولونها وإضاءتها... إلخ... وعندما تقوم بتحريك الفقاعة، سيتولّى Direct3D إحداث التأثيرات التي تشعرك بأنّ المشهد حقيقيّ تماما، وذلك عبر خوارزميّات معقّدة للغاية بالطبع.

ولكن كيف يتمّ تمثيل الخامات؟

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

وسنتعرّف على ذلك بالتفصيل فيما بعد.

التحويلات Transformations:

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

تحويل الإحداثيّات World Ttransform:

افترض أنّ لدينا خمسة مكعّبات في أماكن مختلفة، ونريد عرض لقطة لها معا على الشاشة.. في البداية يمكننا تمثيل هذه المكعّبات بـــ ٤٠ رأسا (٨ لكلّ مكعّب).. بعد هذا سنطلب من Direct3D أن يرسمها لنا.

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

إذن فالأفضل أن ننشئ نسخة واحدة من المكعّب، مركزها نقطة الأصل (٠،٠،٠).. بعد ذلك يمكننا أن نقوم بالتحويلات اللازمة لتغيير موضع هذا المكعّب، مع إمكانيّة تكبيره وتصغيره وتدويره.. بهذه الطريقة يمكننا أن ننشئ أيّ عدد من النسخ من هذا المكعّب بمنتهى البساطة والسرعة.

تحويلات الكاميرا Camera Transforms:

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

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

تحويلات المنظور Projection Transforms:

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

العناصر الهندسيّة Meshes والنماذج Models:

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

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

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

استخدام Direct3D9

مرحبا في VB:

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

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

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

كيف تعدّ جهازك للتعامل مع DirectX9:

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

www.msdn.microsoft.com

وتدخل قسم إنزال البرامج Downloads، ومنه تنزل DirectX9 SDK، وهي حوالي ٢٣٠ ميجا، وهو حجم كبير بالنسبة لسرعات الإنترنت (استغرق منّي عدّة ساعات لإنزاله بتقنية DSL).. تصرّف.. المهمّ أن تحصل على هذا المكوّن.. ولا تتخدع وتنزل المكوّن الخاص بـ VB فقط، فعندما جرّبته لم يعمل!!.. ولكن من المحتمل طبعا أن تكون ميكروسوفت قد تداركت هذا الخطأ.. أنت بالخيار.

ملحوظة١:

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

ملحوظة٢:

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

١- إعداد إطار العمل على جهاز العميل.. إنّ ميكروسوفت تسمح بتداول نسخة مجانية من إطار العمل، يجب أن يتمّ تضمينها ببرنامج الإعداد.
٢- إعداد DirectX9 Runtimes (ليس SDK كلّها) على جهاز العميل.

ملحوظة٣:

بعد أن تعدّ DirectX9، تصفّح المجلّد الذي أنزلته به، وحاول أن تجرّب الأمثلة الموجودة به.. وستجد مجلّدا اسمه Bin به كلّ الملفات التنفيذيّة لهذه الأمثلة.. لو حدثت مشاكل في تشغيل هذه الأمثلة، فاعلم أنّ كارت الشاشة الخاص بك غير مناسب، وعليك تغييره أولا، والحصول على كارت يدعم الرسوم ثلاثيّة الأبعاد.

مشروع تعليميّ: مكعبان دوّران

أعرف أنّك قد مللت من كلّ هذا الكلام النظريّ الجافّ.. تعـــال إذن نبـــدأ رحلة المتعة.

وكبداية، سنحاول أن نتعلّم، عن طريق إنشـــاء مكعّبـــين يـــدوران علـــى الشاشة، كما في الصورة التالية:

ابدأ مشروعا جديدا، وأسمه Basic_D3D9.. أضف لهذا المشروع فئـــة Class وأسمها CSampleGraphicsEngine.vb.

إضافة مرجع لـــ Direct3D:

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

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

Microsoft.DirectX
Microsoft.DirectX.Direct3D
Microsoft.DirectX.Direct3DX

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

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

```
Imports Microsoft.DirectX
Imports Microsoft.DirectX.Direct3D
Imports System.Math
```

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

تعريف المتغيرات:

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

```
Private D3DRoot As Manager
Private D3DDev As Device
Private D3DHelp As D3DX
```

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

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

ولكن ما هو الجهاز Device؟

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

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

بعد ذلك سنقوم بتعريف مصفوفات التحويل Transformation matrices:

```
Private matCube1 As Matrix
Private matCube2 As Matrix
Private matView As Matrix
Private matProj As Matrix
```

وهي تستخدم في عمليات تحويل الإحداثيات المختلفة.

يلي ذلك تعريف الخامات Textures:

```
Private texCube1 As Texture
Private texCube2 As Texture
Private texMenu As Texture
```

ثمّ الأشكال الهندسيّة Geometry:

```
Private vbCube As VertexBuffer
Private vbMenu As VertexBuffer
```

ثمّ الخطوط Fonts:

```
Private fntOut As Font
```

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

وضع القيم الابتدائيّة لـــ Direct3D9

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

الخطوط العامّة:

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

١- تحديد طور العرض display mode والتنسيق Format.

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

٣- ضبط أيّ اختيارات إضافيّة.

٤- إنشاء جهاز Device.

٥- إعداد حالات الرسم Render states.

٦- إعداد مصفوفات التحويلات المبدئيّة.

طور العرض Display Mode والتنسيق Format:

توجد طريقتان في Direct3D لتكوين الأجهزة: فإمّـــا أن تحتـــلّ إحـــدى النوافـــــذ Windowed Mode أو تحتـــــلّ الشاشـــــة كاملـــــة Fullscreen Mode.. أعتقد أنّك تدرك الفارق بين النوعين.

ليس هذا فقط هو ما يهمّنا عند إنشاء الجهاز.. يهمّنا كذلك المساحة التـــي سنستخدمها مـــن الشاشـــة.. هنـــا يجـــب أن تراعـــي طـــور العـــرض

Display Mode، الذي يمثّل مدى دقّة العرض Resolution على الشاشة.. إنّ هذا سيؤثّر كثيرا على الكود الذي سنكتبه.

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

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

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

إنّها تستقبل منك ثلاثة معاملات:

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

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

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

إنّ Direct3D9 يقدّم العديد من التنسيقات الجديدة لمعلومات الألوان.. وعليك أن تقرّر إذا كنت تفضّل استخدام ١٦ خانة ثنائيّة 16 bits أم ٣٢، لحفظ معلومات اللون لكلّ نقطة Pixel.. طبعا استخدام ١٦ خانة أوفر في المساحة وأسرع نوعا في التنفيذ، ولكنّ استخدام ٣٢ خانة أكثر كفاءة على

حساب مساحة الذاكرة (ضعف المساحة).. وكمثال: الشاشة التي مساحتها ١٠٢٤×٧٦٨ بتنسيق ٣٢ خانة لكلّ نقطة تحتاج إلى مساحة ذاكرة تساوي ١٠٢٤×٧٦٨×٣٢ = ٣ ميجا بايت.

ما يهمّنا الآن هو نوعان من التنسيقات:

R5G6B5	في هذا التنسيق يتمّ تخزين مكوّنات لون كلّ نقطة في ١٦ خانة: ٥ لتخزين نسبة لأحمر، و ٦ للأخضر و ٥ للأزرق.
X8R8G8B8	في هذا التنسيق يتمّ تخزين مكوّنات لون كلّ نقطة في ٣٢ خانة: ٨ خانات غير مستخدمة، و ٨ لتخزين نسبة لأحمر، و ٨ للأخضر و ٨ للأزرق.
A1R5G5B5	في هذا التنسيق يتمّ تخزين مكوّنات لون كلّ نقطة في ١٦ خانة: خانة واحدة لتخزين درجة الشفافية، و ٥ لتخزين نسبة لأحمر، و ٥ للأخضر و ٥ للأزرق.

إنّ هناك حوالي ٤٠ تنسيقا مختلفا، ولكنّك تستطيع أن تفهمها كلّها بمجرّد النظر، فهي تحتوي على الحروف A (ويرمز لدرجة الشفافية)، و R (ويرمز للأحمر)، و G (ويرمز للأخضر) و B (ويرمز للأزرق)، و X (ويرمز للخانات غير المستخدمة).. ويلي كلّ حرف عدد الخانات التي يستخدمها.

هذا هو الكود الذي يحدّد التنسيق المناسب:

```
Dim fmt As Format
' اختيار تنسيق مناسب لعمق اللون
Select Case iDepth
    Case 16
        fmt = Format.R5G6B5
    Case 32
        fmt = Format.X8R8G8B8
End Select
```

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

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

```
Dim AdapterInfo As AdapterInformation
Dim D3Dr As Manager
For Each AdapterInfo In D3Dr.Adapters

Next
```

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

```
Dim DispMode As DisplayMode
For Each DispMode In _
        AdapterInfo.SupportedDisplayModes(fmt)

Next
```

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

```
If DispMode.Width = iWidth AndAlso _
            DispMode.Height = iHeight Then
    Return True
End If
```

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

ها هو ذا كود الدالة كاملا:

```
Public Shared Function isDisplayModeOkay(
            iWidth As Integer, iHeight As Integer,
            iDepth As Integer) As Boolean
  Try
      Dim AdapterInfo As AdapterInformation
      Dim DispMode As DisplayMode
      Dim fmt As Format
      Dim D3Dr As Manager
```

```
        ' اختيار تنسيق مناسب لعمق اللون
        Select Case iDepth
            Case 16
                fmt = Format.R5G6B5
            Case 32
                fmt = Format.X8R8G8B8
        End Select
        ' التأكّد من وجود طور عرض مناسب لمساحة العرض وعمق الألوان
        For Each AdapterInfo In D3Dr.Adapters
            For Each DispMode In _
              AdapterInfo.SupportedDisplayModes(fmt)
                If DispMode.Width = iWidth AndAlso _
                  DispMode.Height = iHeight Then
                    Return True
                End If
            Next
        Next
        ' لو وصل التنفيذ لهذا السطر، فهذا يعني عدم العثور على طور عرض مناسب
        Throw New Exception("لم يتمّ العثور على دقّة عرض مناسبة")
    Catch DXErr As DirectXException
        'لتلافي حدوث أي خطإ غير متوقع في DirectX
        Return False
    Catch Err As Exception
        ' لتلافي حدوث أي خطإ آخر غير متوقّع
        Return False
    End Try
End Function
```

المخزن الخلفيّ BackBuffer:

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

أنت تعرف أنّ حركة المجسّم على الشاشة تتمّ عن طريق إعادة رسمه في مواضع متتالية، بسرعة لا تسمح للعين بملاحظة هذه العمليّة (أكثر من ٢٠ مرة في الثانية الواحدة).. ولكي يتمّ ذلك بكفاءة، يجب أن يتمّ رسم المجسّم كاملا أولا في مخزن خفيّ عن عين المستخدم، ثمّ يتمّ إظهار اللقطة على الشاشة كاملة.. بخلاف هذا، سيتمّ رسم المجسّم نقطةً نقطةً أمام المستخدم، وهي عمليّة تتّسم بشيء من البطء، ممّا سيجعل المجسّم يبدو لعين المستخدم وكأنّه يرتعش.

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

دعنا نرى.

حدث إنشاء الفئة New Constructor:

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

ونظرا لأنّ الصيغة الأولى لا تمدّ الفئة بأيّ معلومات حول طريقة العرض، فسنفترض أنّ مستدعيها يريد عرض الرسوم داخل نافذة

Windowed-mode، بينما في الصيغة الثانية سـنفترض أنّ العـرض سيكون في كلّ الشاشة Fullscreen.

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

```
النموذج الذي سيتمّ الرسم عليه ' Private rTarget As Form
طور الشاشة (كاملة/نافذة) ' Private bWindowed As Boolean
سنكتب في هذا المتغيّر معلومات حول طريقة العرض '
Private sDispInfo As String
متغيّر يمثّل نجاح وضع القيم الابتدائيّة أم لا '
Private bInitOkay As Boolean = False
```

ها هي ذي الصيغة الأولى:

```
Public Sub New(
   ByVal Target As System.Windows.Forms.Form)
      Try
            Dim d3dPP As New PresentParameters( )
            d3dPP.Windowed = True
            bWindowed = True
            d3dPP.SwapEffect = SwapEffect.Discard
            d3dPP.BackBufferCount = 1
            d3dPP.BackBufferFormat = _
                  D3DRoot.Adapters(
                  0).CurrentDisplayMode.Format
            d3dPP.BackBufferWidth =
                 Target.ClientSize.Width( )
            d3dPP.BackBufferHeight =
                 Target.ClientSize.Height( )
```

```
            sDispInfo = "[WINDOWED] " + _
                Target.ClientSize.Width.ToString +
                "x" +
                Target.ClientSize.Height.ToString +
                " " +
                d3dPP.BackBufferFormat.ToString( )
            rTarget = Target
            initialiseDevice(CType(Target, Control),
                d3dPP)
        Catch err As Exception
            bInitOkay = False
            Throw New Exception("فشل إعداد محرك الرسوم.")
        End Try
    End Sub
```

إنّ معظم ما فعلناه في هذا الإجراء، هو ملء سجلّ "المعـــاملات الحاليّـــة" PresentParameters بالمعلومات اللازمة، وسنرى فـــيم سنســـتخدمه، وذلك عند شرح الدالة initialiseDevice، التي بدورها تمّ استدعاؤها في هذا الإجراء.

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

```
    Public Sub New(ByVal Target As Form,
                ByVal iWidth As Integer, _
                ByVal iHeight As Integer,
                ByVal iDepth As Integer)
        Try
            Dim d3dPP As New PresentParameters( )
            If Not isDisplayModeOkay(iWidth, iHeight,
                iDepth) Then Throw New Exception(
                    "عرض الشاشة كاملة غير ممكن")
```

```
            bWindowed = False
            d3dPP.BackBufferWidth = iWidth
            d3dPP.BackBufferHeight = iHeight
            d3dPP.BackBufferCount = 1
            d3dPP.SwapEffect = SwapEffect.Copy
            d3dPP.PresentationInterval =
                  PresentInterval.Immediate
            Select Case iDepth
                  Case 16
                        d3dPP.BackBufferFormat =
                              Format.R5G6B5
                  Case 32
                        d3dPP.BackBufferFormat =
                              Format.X8R8G8B8
                  Case Else
                        Throw New Exception(
                              "عمق اللون غير مقبول")
            End Select
            rTarget = Target
            initialiseDevice(CType(Target, Control),
                  d3dPP)
      Catch err As Exception
            bInitOkay = False
            Throw New Exception(
                  "Could not initialize graphics engine")
      End Try
End Sub
```

بخلاف عدد المعاملات، هناك اختلافان جوهريان بين الصيغتين:

الأوّل يتمثّل في عدم استخدام أبعاد النمـــوذج، واســـتخدام المواصـــفات المرسلة كمعاملات بدلا من ذلك، طبعا بعد التأكّد من صلاحيتها للعرض، باستخدام الدالة isDisplayModeOkay.

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

مخزن البعد الثالث Depth Buffer:

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

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

لاحظ أنّ هذا يستلزم عددا كبيرا من عمليات المقارنة، لهذا فإنّ Direct3D يستخدم بعض الخوارزميّات لتوقّع الأوجه الخلفيّة للمجسّمات

(التي لا تواجه الكاميرا) ويتجاهل رسمها، لأنّها في كلّ الأحوال لن تظهر أمام الرائي من هذه الزاوية.
معنى هذا أنّ استخدام مخزن البعد الثالث يسمح لنا برسم المجسّمات دون اعتبار ترتيبها، ففي كلّ الأحوال سيبدو المشهد النهائيّ تماما كما أردناه..
والآن كلّ ما علينا هو استخدام الجملة التالية لتفعيل هذه الإمكانيّة:

D3DDev.RenderState.ZBufferEnable = True

نوع الجهاز Device Type:

قلنا إنّ الكائن Manager هو نقطة انطلاقنا، حيث سنستخدمه لإنشاء الأجهزة التي سنتعامل معها.
شيء أخر نحتاج لإيضاحه، هو ما يختصّ بنوع الجهاز.. إنّ هناك أربعة أنواع، تحدّد الطريقة التي سيتمّ بها إجراء العمليّات على نقاط الصورة:

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

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

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

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

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

```
If D3DCaps.DeviceCaps.
  SupportsHardwareTransformAndLight Then
```

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

```
DevCreate = CreateFlags.HardwareVertexProcessing Or
            CreateFlags.MultiThreaded
```

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

```
DevCreate = CreateFlags.SoftwareVertexProcessing Or
            CreateFlags.MultiThreaded
```

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

```
D3DDev = New Device(0, DeviceType.Hardware,
         Target, DevCreate, win)
```

دالة وضع القيم الابتدائيّة:

سنعرّف الآن المزيد من المتغيّرات الخاصة على مستوى الفئة:

```
حجم الخط الذي سيتم رسم النصوص به '
Private iFontSize As Integer
Private lastFrameUpdate As Int32
Private sDevInfo As String ' اسم كارت الشاشة
```

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

```
Private Sub initialiseDevice(ByVal Target As Control,
        ByVal win As PresentParameters)
    Try
        Dim D3DCaps As Caps
        Dim DevCreate As Integer

        '1. التحقق من مخزن البعد الثالث، وتوصيفه
        If D3DRoot.CheckDepthStencilMatch(0, _
                DeviceType.Hardware,
                win.BackBufferFormat, _
                win.BackBufferFormat,
                DepthFormat.D16) Then
            win.AutoDepthStencilFormat =
                DepthFormat.D16
            win.EnableAutoDepthStencil = True
        Else
            Throw New Exception("خطأ")
        End If
```

```
'2. اختبار قدرات الجهاز
D3DCaps = D3DRoot.GetDeviceCaps(0, _
                DeviceType.Hardware)
' الاستفادة من إمكانيات مكونات الجهاز في التحويلات والإضاءة
If D3DCaps.DeviceCaps.
 SupportsHardwareTransformAndLight  Then
     DevCreate = _
     CreateFlags.HardwareVertexProcessing Or
      CreateFlags.MultiThreaded
Else
     DevCreate = _
     CreateFlags.SoftwareVertexProcessing Or _
    CreateFlags.MultiThreaded
End If
'3. محاولة إنشاء واجهة الجهاز
D3DDev = New Device(0,
       DeviceType.Hardware, Target,
       DevCreate, win)
If D3DDev Is Nothing Then
   Throw New Exception("…")
End If
'4. ضبط حالة الرسم والمعاملات الأخرى
' استخدام مخزن للبعد الثالث (العمق)
D3DDev.RenderState.ZBufferEnable = True
' لن نستخدم الإضاءة في هذا المثال
D3DDev.RenderState.Lighting = False
' السماح بشفافية الخامات
D3DDev.RenderState.SourceBlend =
       Blend.SourceAlpha
```

```
D3DDev.RenderState.DestinationBlend =
    Blend.InvSourceAlpha
' إعداد خليط الخامات
D3DDev.SamplerState(0).MinFilter =
    TextureFilter.Linear
D3DDev.SamplerState(0).MagFilter =
    TextureFilter.Linear
'5. تحميل الخامات
loadTextures(win.BackBufferFormat)
'6. إنشاء الرسوم الضرورية
loadGeometry( )
'7. إعداد المصفوفات
'view matrix: تصف خصائص الكاميرا
D3DDev.Transform.View =
        Matrix.LookAtLH(New Vector3( _
        0, 0, -15), New Vector3(0, 0, 0), _
        New Vector3(0, 1, 0))
'projection matrix: تصف خصائص عدسات الكاميرا
D3DDev.Transform.Projection = _
        Matrix.PerspectiveFovLH(Math.PI / 4,
        4 / 3, 1, 100)
'world matrix:
' تصف إحداثيات الأشكال وما سيتمّ عليها من تحويلات
D3DDev.Transform.World = Matrix.Identity( )
'8. sort out fonts for on-screen rendering
iFontSize = 11
fntOut = New Font(D3DDev, _
        New Drawing.Font("Arial", iFontSize,
        FontStyle.Bold))
```

```
            '9. ضبط المتغيّرات العامة
            lastFrameUpdate = Environment.TickCount( )
            ' الاحتفاظ باسم كارت الشاشة في متغيّر
            sDevInfo =
                D3DRoot.Adapters(
                0).Information.Description
            bInitOkay = True
        Catch err As Exception
            bInitOkay = False
            Throw New Exception("فشل إعداد محرك الرسوم")
        End Try
    End Sub
```

تكوين المجسّمات في الذاكرة

كيفية تخزين المجسّمات:

يمكنك أن تحفظ رءوس المجسّمات vertices بإحدى طريقتين:

١- مخازن الرءوس Vertex Buffers:

وهي بمثابة مصفوفة يوفرّها لك Diurect3D لمعالجة الرسوم بكفاءة عالية، حيث يتحكّم في كيفيّة تخزين الرسوم (إمّا في ذاكرة الكمبيوتر أو ذاكرة كارت الشاشة).. وحتّى تحصل على أفضل أداء من هذه الطريقة، استخدم مخزن الرءوس Vertex Buffer لتخزين عدد من الرءوس ما بينَ ١٠٠ و ٥٠٠٠.. فإذا زاد العدد عن ذلك، فاقسمه على أكثر من مخزن.. وإذا قلّ العدد عن ذلك، فضع رءوس أكثر من مجسّم معا في نفس المخزن.

٢- باستخدام المصفوفات التقليديّة:

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

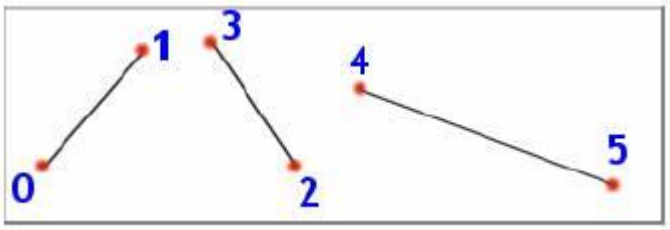
كيفيّة رسم المجسّمات:

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

إنّ هناك ٦ طرق للتعامل مع هذه الرءوس:

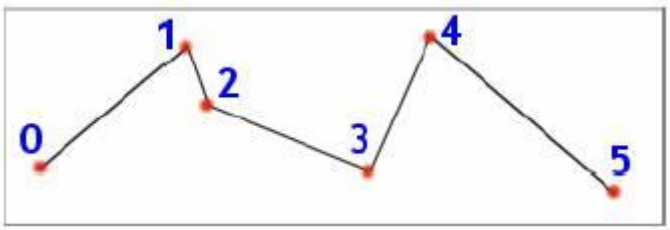
١ - قائمة الخطوط Line List:

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



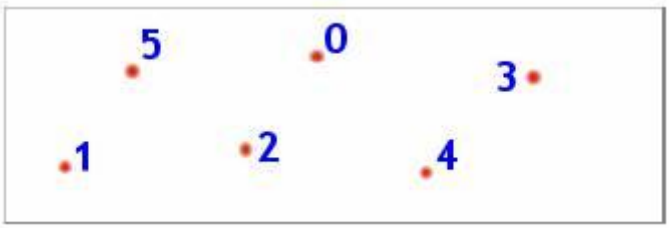
٢ - شريط الخطوط Line Strip:

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



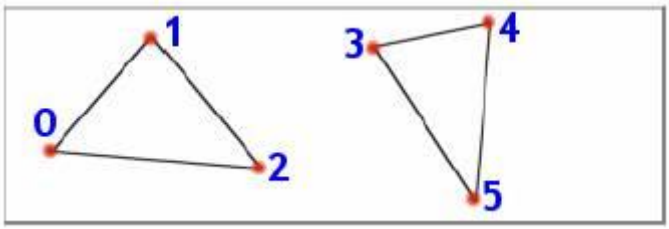
٣ - قائمة النقاط Point List:

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



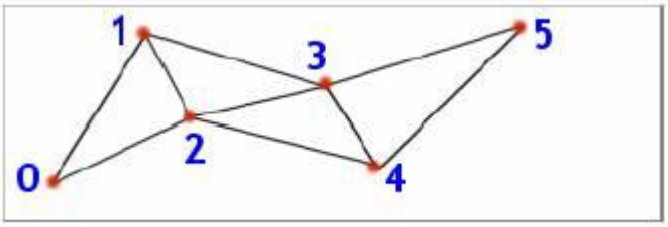
٤ - قائمة المثلثات Triangle List:

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



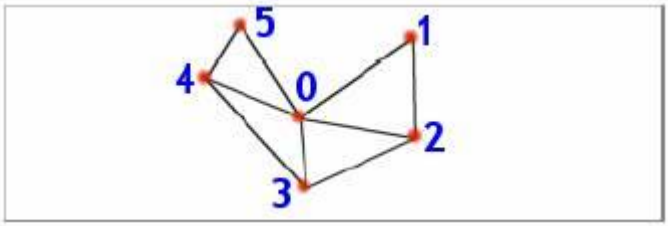
٥ - شريط المثلثات Triangle Strip:

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



٦ - مروحة المثلات Triangle Fan:

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



تحميل المجسمات:

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

```
Private Sub loadGeometry( )
    Try
        ' تحميل الرسوم ثنائية البعد (قائمة الأوامر) في الرءوس
        '[----------------------------------]
        vbMenu = New VertexBuffer(GetType( _
        CustomVertex.TransformedTextured), 4,
           D3DDev, 0, _
           CustomVertex.TransformedTextured.Format,
           Pool.Default)
```

```
Dim v( ) As _
        CustomVertex.TransformedTextured = _
        CType(vbMenu.Lock(0, 0), _
        CustomVertex.TransformedTextured( ))
Const iMBWidth As Integer = 158
Const iMBHeight As Integer = 93
v(0) = New _
        CustomVertex.TransformedTextured(0,
        86, 0, 1, 0, 2 / 127)
v(1) = New _
        CustomVertex.TransformedTextured(
         iMBWidth, 86, 0, 1,
         iMBWidth / 255, 2 / 127)
v(2) = New
        CustomVertex.TransformedTextured(0, _
        86 + iMBHeight, 0, 1, 0, iMBHeight / 127)
v(3) = New
        CustomVertex.TransformedTextured( _
        iMBWidth, 86 + iMBHeight, 0, 1, _
        iMBWidth / 255, iMBHeight / 127)
vbMenu.Unlock( )
'[----------------------------------]
' تحميل الرسوم المجسمة في الرءوس '
'[----------------------------------]
vbCube = New VertexBuffer( _
GetType(CustomVertex.PositionTextured), _
        36, D3DDev, 0, _
        CustomVertex.PositionTextured.Format, _
        Pool.Managed)
```

```
Dim vCube As _
        CustomVertex.PositionTextured( ) = _
        CType(vbCube.Lock(0, 0), _
        CustomVertex.PositionTextured( ))
'2a. إنشاء وجه المكعب العلويّ
vCube(0) = New
        CustomVertex.PositionTextured(-0.5, 0.5, _
        0.5, 0, 0)
vCube(1) = New
        CustomVertex.PositionTextured(0.5, 0.5, _
        0.5, 1, 0)
vCube(2) = New
        CustomVertex.PositionTextured(-0.5, 0.5, _
        -0.5, 0, 1)
 vCube(3) = New
         CustomVertex.PositionTextured(0.5, 0.5, _
         -0.5, 1, 1)
vCube(4) = New
         CustomVertex.PositionTextured(-0.5, 0.5,
         -0.5, 0, 1)
vCube(5) = New
         CustomVertex.PositionTextured(0.5, 0.5, _
         0.5, 1, 0)
'2b. إنشاء وجه المكعب السفليّ
vCube(6) = New
         CustomVertex.PositionTextured(-0.5, -0.5,
         -0.5, 1, 0)
vCube(7) = New
         CustomVertex.PositionTextured(0.5, -0.5,
         0.5, 0, 1)
```

```
vCube(8) = New
        CustomVertex.PositionTextured(-0.5, -0.5,
        0.5, 0, 0)
vCube(9) = New
       CustomVertex.PositionTextured(0.5, -0.5, _
       0.5, 0, 1)
vCube(10) = New
       CustomVertex.PositionTextured(-0.5, _
       -0.5, -0.5, 1, 0)
vCube(11) = New
       CustomVertex.PositionTextured(0.5, -0.5, _
       -0.5, 1, 1)
```

'2c. إنشاء وجه المكعب الأيسر

```
vCube(12) = New
       CustomVertex.PositionTextured(-0.5, 0.5,
       -0.5, 0, 0)
vCube(13) = New
        CustomVertex.PositionTextured(-0.5, _
       -0.5, -0.5, 1, 0)
vCube(14) = New
       CustomVertex.PositionTextured(-0.5, 0.5,
       0.5, 0, 1)
vCube(15) = New
       CustomVertex.PositionTextured(-0.5, 0.5,
       0.5, 0, 1)
vCube(16) = New
       CustomVertex.PositionTextured(-0.5, _
       -0.5, -0.5, 1, 0)
vCube(17) = New
       CustomVertex.PositionTextured(-0.5, _
       -0.5, 0.5, 1, 1)
```

```
'2d. إنشاء وجه المكعب الأيمن
vCube(18) = New
        CustomVertex.PositionTextured(0.5, 0.5, _
        -0.5, 0, 0)
vCube(19) = New
        CustomVertex.PositionTextured(0.5, 0.5, _
        0.5, 1, 0)
vCube(20) = New
        CustomVertex.PositionTextured(0.5, -0.5,
        -0.5, 0, 1)
vCube(21) = New
        CustomVertex.PositionTextured(0.5, -0.5,
        0.5, 1, 1)
vCube(22) = New
         CustomVertex.PositionTextured(0.5, -0.5,
        -0.5, 0, 1)
vCube(23) = New
         CustomVertex.PositionTextured(0.5, 0.5,
        0.5, 1, 0)
'2e. إنشاء وجه المكعب الخلفيّ
vCube(24) = New
         CustomVertex.PositionTextured(-0.5, 0.5,
        -0.5, 0, 0)
vCube(25) = New
         CustomVertex.PositionTextured(0.5, 0.5,
        -0.5, 1, 0)
vCube(26) = New
         CustomVertex.PositionTextured(-0.5, _
        -0.5, -0.5, 0, 1)
vCube(27) = New
         CustomVertex.PositionTextured(0.5, -0.5,
        -0.5, 1, 1)
```

```
        vCube(28) = New
                CustomVertex.PositionTextured(-0.5, _
                -0.5, -0.5, 0, 1)
        vCube(29) = New
                CustomVertex.PositionTextured(0.5, 0.5,
                -0.5, 1, 0)
        '2f. إنشاء وجه المكعب الأماميّ
        vCube(30) = New
                CustomVertex.PositionTextured(0.5, 0.5,
                0.5, 1, 0)
        vCube(31) = New
                CustomVertex.PositionTextured(-0.5, 0.5,
                0.5, 0, 0)
        vCube(32) = New
                CustomVertex.PositionTextured(-0.5, _
                -0.5, 0.5, 0, 1)
        vCube(33) = New
                CustomVertex.PositionTextured(-0.5, _
                -0.5, 0.5, 0, 1)
        vCube(34) = New
                CustomVertex.PositionTextured(0.5, -0.5,
                0.5, 1, 1)
        vCube(35) = New
                CustomVertex.PositionTextured(0.5, 0.5, _
                0.5, 1, 0)
        vbCube.Unlock( )
    Catch err As Exception
        MsgBox("loadGeometry( ): " + Chr(13) +
                Chr(13) + err.ToString( ))
        Throw New Exception("Err in loadGeometry")
    End Try
End Sub
```

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

في كلتا الحالتين سنبدأ بحجز مخزن الرءوس VertexBuffer:

```
vbCube = New VertexBuffer(GetType(
    CustomVertex.PositionTextured), 36, D3DDev, 0,
    CustomVertex.PositionTextured.Format,
    Pool.Managed)
```

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

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

```
Dim vCube( ) As CustomVertex.PositionTextured = _
    CType(vbCube.Lock(0, 0),
    CustomVertex.PositionTextured( ))
```

الآن يمكننا التعامل مع الخانات من ٠ إلى ٣٥ (٣٦ خانة كما عرّفناها) في المصفوفة (المخزن) vCube.

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

لاحظ أنّنا في هذا المثال استخدمنا طريقة قائمة المثلثات Triangle-List

للتوصيل بين رءوس المكعّب.

آخر خطوة في هذه العمليّة، هي تحرير المخـــزن الـــذي أغلقنـــاه أمـــام التطبيقات الأخرى.. إنّ هذه الخطوة ضروريّة، لأنّـــك Direct3D لـــو حاول أن يرسم المكعّب على الشاشة في حين أنّ مخزن الرءوس مغلـــق، فسيقوم بإطلاق استثناء ولن ينفّذ العمليّة:

vbCube.Unlock( )

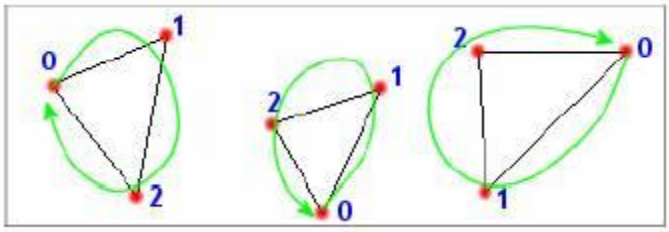
ترتيب الرءوس في المثلث:

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

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

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

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



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

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

حسنا.. إنّ قتل الأوجه الخلفيّة Culling يوفّر ٤٠% من الوقت اللازم لرسم المجسّم بدون استخدام هذه التقنية.. فإذا لاحظت أنّ وجها واحدا فقط (من وجوه المكعّب الستّة) هو الذي يسهل تحديد كونه وجها خلفيّا، لأنّه يقع في المستوى (س ــ ص)، فلا بدّ أنّك ستدرك السرّ وراء احتياج Direct3D لتحديد اتجاه الإزالة.. إنّ عليه أن يزيل ٣ أوجه من خمسة وجوه، ويحتاج منك لاختيار وسيلة سريعة لتحديدها.

والآن تعال نواصل مشروعنا.

الخامات Textures

الآن نتحدّث بمزيد من العمق عن الخامات:

تنسيقات الخامات:

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

ولكي تحصل على أفضل أداء للخامة، اجعل مقاييسها ثنائيّة (أعداد مرفوعة للأس ٢)، مثل ١٢٨×١٢٨، ٢٥٦×٢٥٦، ٥١٢×١٠٢٤.... إلخ.. إنّ هذا سيسهّل على معظم أنواع كروت الشاشة تسريع التعامل مع هذه الخامات.

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

لكن يجب عليك أن تتأكّد أولا من المدى الذي يستطيع فيه كارت الشاشة التعامل مع الخامات، حيث إنّ معظم الكروت تتعامل مع خامات مساحتها أصغر من أو تساوي ١٠٢٤×١٠٢٤، وهو كاف بالفعل في معظم الحالات.

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

وهناك امتداد آخر هو DDS (سطح الرسم المباشر Direct Draw Surface)، يتمّ فيه حفظ الصورة بنفس التنسيق الذي يتمّ تمثيلها به في الذاكرة بواسطة كارت الشاشة.. إنّ هذا يعني تحميل الصورة من الملفّ بسرعة فائقة، كما يمنح مصمّم الصورة القدرة على

تحديد عدد الخانات التي تمثّل درجة الشفافية كما يريد، كأن يجعلها خانتين Bits 2 بدلا من ٨، دون أن يضيّع Direct3D وقتا طويلا في التحويلات.

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

وعند تحميل الخامة في ذاكرة كارت الشاشة، يتمّ حفظها بتنسيق معيّن.. ويمنحنا Direct3D ٤٠ تنسيقا مختلفا، أسماؤها على شاكلة X8R8G8B8، وقد تعرّفنا من قبل على ما تعنيه الحروف R و G و B، وما تعنيه الأرقام التالية لكلّ منها.

ما يجب أن تلاحظه هنا، هو أنّ هناك مساحة مخصّصة من كارت الشاشة لتحميل الخامات (لو كان كارت الشاشة ٣٢ ميجا، فستتوفر مساحة من ٢٠ إلى ٢٥ ميجا للخامات)، وعليك ألا تتجاوز هذه المساحة، وذلك بتحميل الخامات الضروريّة في الوقت المناسب، والتخلّص من الخامات التي انتهى دورها.

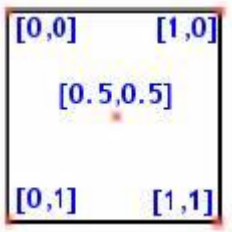
إحداثيّات الخامات Texture Coordinate:

لكلّ رأس Vertex موقع ولون وإضاءة وإحداثيّات خامة.

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

إنّ هذا هو السبب الذي جعلنا نرسل خمس إحداثيّات أثناء ملء مصـــفوفة الرءوس في دالة تحميل الرسوم: ٣ إحداثيّات الموضع، وإحداثيّان الخامة. مع ملاحظ أنّ وحدة قياس الخامة تختلف عن وحدة قياس الموضع (يقاس بالنقطة Pixel)، حيث تقاس الخامات بأرقام عشريّة تنحصر بـــين ٠,٠ و ١,٠، وأيّ أعداد خارج هذا النطاق لا يتمّ تمثيلها.

والصورة والجدول التاليين يوضّحان لك إحداثيّات بعض المواضـــع فـــي الخامة:



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

إنّ لعدم استخدام الوحدات التقليديّة حكمة.. افترض أنّ المثلّث الذي ســيتمّ لصق الخامة عليه يتحرّك بعيدا عن الكاميرا.. معنـــى هـــذا أن حجمـــه الظاهريّ سيقلّ.. في هذه الحالة لا جدوى من تحميل خامـــة ذات كفـــاءة

عالية على مثلّث ضئيل المساحة.. لهذا فإنّ Drect3D يُنتج نسخا مختلفة من الخامة بدقّة مختلفة (٢٥٦×٢٥٦، ١٢٨×١٢٨، ٦٤×٦٤.... ٢×٢، ١×١)، ويختار مستوى الدقّة تبعا لحجم المثلّث وبعده عن الكاميرا.. لهذا فإنّ التعامل بإحداثيّات نسبيّة للخامة أفضل من التعامل مع إحداثيّات الشاشة، حيث سيقوم Direct3D بإجراء التحويلات المناسبة.

كما أنّ هذا يسمح لك أيضا بتغيير مساحة الخامة دون أن تغيّر الكود.. افترض أنّ لديك خامة ١٠٢٤×١٠٢٤ ولم يستطع كارت الشاشة التعامل معها.. في هذه الحالة يمكنك تحميلها بأبعاد أصغر، دون أن تخشى من حدوث مشكلة في تنفيذ الكود، فالإحداثيّات التي تستخدمها نسبيّة وليست حقيقيّة.

شيء آخر يجب أن تعرفه، هو أنّه من الممكن أن يكون للمثلّث الواحد أكثر من خامة (قد يصل الأمر إلى ٨ خامات للمثلّث)، وطبعا يمكن ألا تستخدم الخامات أساسا.

تحميل الخامات:

نصل الآن لكود الدالة loadTextures، وهو بسيط للغاية.

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

ها هو ذا الكود:

```
Sub loadTextures(ByVal AdapterFmt As Format)
    ' تحميل خامة القائمة
    ' يجب التأكّد أولا من صلاحية التنسيق للتعامل مع كارت الشاشة
    If D3DRoot.CheckDeviceFormat(0,
       DeviceType.Hardware, AdapterFmt, 0,
       ResourceType.Textures,
       Format.A8R8G8B8) Then
          texMenu = TextureLoader.FromFile(
                D3DDev, Application.StartupPath +
                "\menubar.bmp", 256, _
                128, 1, 0, Format.A8R8G8B8, _
                Pool.Managed, Filter.None, _
                Filter.None,
                Drawing.Color.Magenta.ToArgb( ))
    ElseIf D3DRoot.CheckDeviceFormat(0, _
                DeviceType.Hardware, AdapterFmt,
                0, ResourceType.Textures,
                Format.A1R5G5B5) Then
          texMenu = TextureLoader.FromFile(
                D3DDev, Application.StartupPath +
                "\menubar.bmp", 256, 128, 1, 0,
                Format.A1R5G5B5, Pool.Managed, _
                Filter.None, Filter.None, _
                Drawing.Color.Magenta.ToArgb( ))
    Else
          Throw New Exception("خطأ")
    End If
    ' تحميل خامة المكعّبين
```

```
        If D3DRoot.CheckDeviceFormat(0,
           DeviceType.Hardware, AdapterFmt, 0,
           ResourceType.Textures, _
           Format.X8R8G8B8) Then
              texCube1 = TextureLoader.FromFile(
                    D3DDev, Application.StartupPath +
                    "\cube1.bmp", 256, 256, 1, 0,
                    Format.X8R8G8B8, Pool.Managed, _
                    Filter.Linear, Filter.Linear, 0)
              texCube2 = TextureLoader.FromFile(
                    D3DDev, Application.StartupPath +
                    "\cube2.bmp", _
                    256, 256, 1, 0, Format.X8R8G8B8,
                    Pool.Managed, _
                    Filter.Linear, Filter.Linear, 0)
        ElseIf D3DRoot.CheckDeviceFormat(0,
                  DeviceType.Hardware, AdapterFmt, 0,
                  ResourceType.Textures, _
                  Format.R5G6B5) Then
              texCube1 = TextureLoader.FromFile(
                    D3DDev, Application.StartupPath +
                    "\cube1.bmp", 256, 256, 1, 0,
                    Format.R5G6B5, Pool.Managed, _
                    Filter.Linear, Filter.Linear, 0)
              texCube2 = TextureLoader.FromFile(
                    D3DDev, Application.StartupPath +
                    "\cube2.bmp", 256, 256, 1, 0,
                    Format.R5G6B5, Pool.Managed, _
                    Filter.Linear, Filter.Linear, 0)
        Else
              Throw New Exception("…")
        End If
End Sub
```

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

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

نوع الجهاز: ونحن نستخدم في هذا المشروع النوع Hardware.

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

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

نوع المصدر: وهو هنا ResourceType.Textures.

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

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

مصفوفات التحويل Transformation Matrices:

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

عامّة هذه هي أهمّ العمليّات التي يمكن إجراؤها على المجسّمات:

١- تغيير الحجم Scale.

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

٣- تغيير الموضع Translation.

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

لحسن الحظّ لا.. إنّ لدينا مكتبة رياضيّات Math library يمدّنا بها Direct3DX، وهي ستقوم بمعظم العمل المعقد بنفسها.. إنّ سجلّ المصفوفة Matrix structure يمدّنا بأكثر من ٢٥ دالة تسهّل علينا التعامل مع مصفوفات التحويل.. والجدول التالي يوضّح لك سبعة من أهمّ هذه الدوال:

Multiply	لضرب مصفوفتين معا (لتكوين مصفوفة جديدة تدمج عملتي التحويل معا).
Identity	مصفوفة الوحدة (كلّ عناصرها أصفار ما عدا قطرها،

	فكلّه آحاد).. عند ضرب هذه المصفوفة في أيّ مصفوفة أخرى لا يحدث أيّ تأثير (مثلما تضرب الواحد في أيّ عدد).
RotateX	لتدوير المجسّم حول محور السينات.
RotateY	لتدوير المجسّم حول محور الصادات.
RotateZ	لتدوير المجسّم حول محور العينات.
Scaling	تغيّر حجم المجسّم بنسبة معيّنة (ترسلها كمعامل).
Translation	لتغيير موضع المجسّم.

انظر كيف نغيّر حجم المكعّب بالنسبة لجميع المحاور:

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

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

```
Private matCube1 As Matrix
```

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

```
Private M As Matrix
M = Matrix.Scaling(3, 3, 3)
matCube1 = Matrix.Multiply(matCube1, M)
```

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

إذن كيف يمكن إحداث تأثيرات مختلفة على المجسّمات؟

بسيطة: كوّن مصفوفة التحويل الأولى وارسم المجسّم الأوّل.. هذا يعني أنّه سيتأثّر بهذه التحويلات.. بعد ذلك كوّن مصفوفة التحويل الثانية وارسم

المجسّم الثاني.. سيتأثّر المجسّم الثاني فقط بـــالتحويلات الجديـــدة، بينمـــا المجسّم الأوّل ثابت كما هو.. ويمكن أن ترسم أكثر من مجسّم معا بـــنفس مصفوفة التحويل.. بعد أن تنتهي ستحصل على اللقطة التي تريدها، وبها كلّ مجسّم في الوضع المطلوب.

والآن تعال نكمل مشروعنا، لنرى كيف سنرسم المكعّبين ونحرّكهما.

تحديث الإطار Frame Update

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

رسم الإطار:

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

```
Private cube1Angle As Single
Private cube2Angle As Single
Private Const cube1Speed As Single = 50.0F
Private Const cube2Speed As Single = 75.0F
Private Const cube1Size As Single = 2.0F
Private Const cube2Size As Single = 4.0F
Private iLastFPSCheck As Int32
Private Const iFPSProfileSpeed As Integer = 200 'ms
Private iFrameRate As Integer
Private iCurrCnt As Integer
```

والآن، ها هو ذا كود الإجراء:

```
Public Sub oneFrameUpdate( )
    If Not bInitOkay Then
       Throw New Exception("…")
    End If
    Dim matTmp As Matrix
```

```
'حساب زوايا الدوران للمكعبين
cube1Angle += ((Environment.TickCount( ) –
      lastFrameUpdate) / 1000) * cube1Speed
cube2Angle += ((Environment.TickCount( ) –
      lastFrameUpdate) / 1000) * cube2Speed
lastFrameUpdate = Environment.TickCount( )
'مصفوفة التحويل للمكعّب الأوّل
matCube1 = Matrix.Identity( ) ' مصفوفة الوحدة.. لا تأثير
' تغيير الحجم
matCube1 = Matrix.Multiply(matCube1,
            Matrix.Scaling(3, 3, 3))
' تدوير حول المحور السيني
matCube1 = Matrix.Multiply(matCube1, _
      Matrix.RotationX(cube1Angle *
      (Math.PI / 180)))
' تدوير حول المحور الصادي
matCube1 = Matrix.Multiply(matCube1, _
      Matrix.RotationY(cube1Angle *
      (Math.PI / 180)))
'مصفوفة التحويل للمكعّب الثاني
matCube2 = Matrix.Identity( )
' تغيير الحجم
matCube2 = Matrix.Multiply(matCube2,
     Matrix.Scaling(4, 5, 0.75))
' تغيير موضع المكعّب
matCube2 = Matrix.Multiply(matCube2, _
      Matrix.Translation(-8, -4, 0))
```

```
        ' تدوير المكعّب حول محور العينات
        matCube2 = Matrix.Multiply(matCube2, _
            Matrix.RotationZ(cube2Angle *
            (Math.PI / 180)))
        ' حساب الإطار الحاليّ
        If (Environment.TickCount( ) –
           iLastFPSCheck >= iFPSProfileSpeed) Then
            iLastFPSCheck = Environment.TickCount( )
            iFrameRate = iCurrCnt *
                (1000 / iFPSProfileSpeed)
            iCurrCnt = 0
        End If
        iCurrCnt += 1
End Sub
```

والفكرة في هذا الكود، هي تغيير زاوية دوران المكعب بمقدار ثابت في كلّ ثانية (٥٠ درجة في الثانية مثلا).. إنّ هذه الطريقة تضمن أداء ثابتا، فمهما زاد أو قلّ عدد الإطارات التي يتمّ رسمها في الثانية، فإنّ دورانا مقداره ٥٠ درجة فقط هو الذي يحدث للمجسّم.. أمّا لو كنّا نعتمد على استخدام سرعة معيّنة للدوران (عدد الإطارات في الثانية)، وعلى أساس ذلك يتحدّد مقدار التغيير في زاوية الدوران، فإنّ هذا قد يؤدّي لعدم استقرار الحركة، فقد تكون سريعة جدّا على الكمبيوترات السريعة، وقد تكون بطيئة جدّا ومتقطّعة إذا كان هناك ما يعطّل الكمبيوتر أو كانت سرعته بطيئة.

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

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

لاحظ أنّنا نحتفظ بآخر وقت حدّثنا فيه الإطار في المتغيّر lastFrameUpdate، وبطرحه من الوقت الحالي يمكن معرفة الوقت المنقضي.. وطبعا نقسم على ١٠٠٠ حتّى نقرّب الناتج لأقرب ثانية.

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

cube1Angle += ((Environment.TickCount( ) –
lastFrameUpdate) / 1000) * cube1Speed

المعادلة تقول ببساطة، إنّ الزاوية الجديدة تزيد عن الزاوية القديمة (لاحظ علامة +=)، بمقدار الوقت الذي انقضى (بالثانية) منذ رسم الإطار السابق، مضروبا في سرعة الدوران (معبّرا عنها بثابت، ليسهل عليك تغييرها في أيّ لحظة.. هذتا الثابت هنا يساوي ٥٠).

إنّ هذا معناه أنّ دوران المجسّم سيكون مستقّرا وبسرعة ثابتة، فلو حدث شيء عطّل الكمبيوتر لثلاث ثوان مثلا، فستزيد زاوية الدوران بمقدار ٣×٥٠ = ١٥٠ مرّة واحدة، بحيث يبدو وكأنّ المجسّم كان يواصل حركته بطريقة طبيعيّة.. طبعا ليس من الشائع أن ينشغل الكمبيوتر عن تنفيذ برنامجنا لثلاث ثوان دفعة واحدة، والأحرى أن نتوقّع أن يتعطّل لأجزاء من مئات الأجزاء من الثانية على أسوأ تقدير!

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

lastFrameUpdate = Environment.TickCount( )

الآن علينا أن نجرى التحويلات على المكعّبين.. تعال نرى مـــا ســـيحدث للمكعّب الأوّل:

١- تحجيم هذا المكعّب ليصبح ٣×٣×٣.. تذكّر أنّ المكعّب الأصليّ الذي أنشأناه كان ١×١×١.

٢- تدوير هذا المكعّب حول محور السينات ثمّ محور الصادات بمقـــدار زاوية الدوران التي حسبناها من قبل.. لاحظ أنّنا قمنا بتحويل الزاوية إلى القياس الدائريّ وذلك بالضرب في النسبة التقريبيّـــة ط والقســـمة على ١٨٠:

```
cube1Angle * (Math.PI / 180)
```

لاحظ كذلك أنّ زاوية الدوران تبدأ من صفر حتّى ٣٦٠ (دورة كاملة) ثمّ تعود إلى الصفر مرّة أخرى.. ولن نحتاج للتدخّل في هذا، فهو من أبسط قواعد الرياضيّات و Direct3D يفهمه!

لاحظ أنّنا لن ننقل هذا المكعّب من موضعه في مركـــز الشاشـــة، حيـــث سنكتفي بتدويره حول مركزه.

تعال نرى ما سنفعله بالمكعّب الثاني:

١- سنغيّر أبعاد المكعّب.. ونحن هنا لن نتركـــه مكعّبـــا، بـــل ســـنحوّله لمتوازي مستطيلات، طوله وعرضه وارتفاعه مختلفة عن بعضها.

٢- بعد هذا سننقل هذا المجسّم بعيدا عن مركز المكعّب الأصليّ بمقـــدار (-٨، -٤، ٠).

٣- ثمّ سندير هذا المجسّم حول محور العينات Z axis بمقـــدار زاويـــة الدوران التي حسبناها له من قبل.. إنّ هذا سيجعل المجسّم يدور حول نقطة الأصل.. لماذا؟.. لأنّنا نقلناه من موضعه أولا.. إنّ الدوران سيتمّ حول المحور عين، وليس حور مركز المجسّم نفسه.. ولـــو أردت أن

تجعل المجسّم يدور حول محوره هو، فقم بعمليّة الدوران أولا قبل نقل المجسّم من موضعه.. فهمت الآن فائدة الترتيب؟

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

اعتبارات يجب مراعاتها عند الرسم:

قبل أن نقوم بالخطوة الأخيرة، يجب أن تضع في اعتبارك أنّ أكثر من ٩٩% من وقت تنفيذ برامج الرسوم يضيع في رسم المجسّمات، وليس في تحميل الصور والخامات ووضع القيم الابتدائيّة.. لهذا فإنّ خبرتك في البرمجة ستتمرّكز حول كيفيّة تحسين أداء تطبيقك، بحيث لا تضيّع وقتا أطول من اللازم في رسم المجسّمات.. وفيما يلي بعض المطبّات التي يجب أن تتلافاها:

١ - الرسم زيادة عن الحدّ Overdraw:

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

٢ - المجسّمات شبه الشفّافة:

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

٣ - الرسم بالقوّة الشرسة Brute-force Rendering:

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

رسم المكعّبين على الشاشة:

آخر خطوة أمامنا الآن هي أن نرسم المكعّبين والقائمة على الشاشة (رسم الإطار).. إنّ ذلك سيتمّ عن طريق الإجراء oneFrameRender. لاحظ أنّ هذا الإجراء هو المكان الذي يجب أن تكتب فيه حلولا للمشاكل التي ناقشناها سابقا، مثل التخلّص من المجسّمات الموجودة خارج نطاق الرؤية.

ها هو ذا الهيكل العامّ للكود:

```
Public Sub oneFrameRender( )
    If Not bInitOkay Then Throw New Exception("…")
    ' محو الإطار السابق ومخزن البعد الثالث
    D3DDev.Clear(ClearFlags.Target Or
        ClearFlags.ZBuffer, _
        Drawing.Color.FromArgb(255, 0, 0, 64),
        1.0F, 0)
    ' بداية الإطار
    D3DDev.BeginScene( )
    '*****************************
    ' ضع كود الرسم هنا
    '*****************************
    ' إنهاء الإطار
    D3DDev.EndScene( )
    ' تنفيذ رسم المجسّمات
    D3DDev.Present( )
End Sub
```

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

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

```
تحديد مصدر الرسم عن طريق تمرير مصفوفة رءوس المكعّب '
D3DDev.SetStreamSource(0, vbCube, 0)
تحديد نوع تنسيق البيانات في مصفوفة الرءوس '
D3DDev.VertexFormat =
    CustomVertex.PositionTextured.Format
```

وأخيرا سنرسم المكعّب الأوّل:

```
If bRenderTextures Then
    رسم الخامة على أوجه المكعّب '
    D3DDev.SetTexture(0, texCube1)
Else
    عدم رسم الخامة على أوجه المكعّب '
    D3DDev.SetTexture(0, Nothing)
End If
وضع مصفوفة التحويل الخاصة بالمكعّب الأوّل في مصفوفة الموضع '
D3DDev.Transform.World = matCube1
تحديد طريقة الرسم (قائمة مثلثات).. '
يتكوّن المكعّب من ١٢ مثلثا سنبدأ رسمها من أولها (رقم ٠) '
D3DDev.DrawPrimitives(PrimitiveType.TriangleList,
    0, 12)
```

لاحظ أنّ bRenderTextures هو متغيّر خاصّ بالفئة يحتوي على قيمة الخاصيّة useTextures، وهي خاصيّة يجب أن تضيفها للفئة لتسمح

للمبرمج الذي يستخدمها بتحديد إذا ما كان يرغب في استخدام الخامات أم لا (خاصيّة منطقيّة Boolean).. أعتقد أنّك تستطيع أن تكتب كود هــذه الخاصيّة، فهو كود تقليديّ لا علاقة له بـــ DirectX علـــى الإطـــلاق.. (راجع الكود في التطبيق المرفق بهذا الفصل).

وسيتمّ رسم المكعّب الثاني بنفس الطريقـــة، مـــع اســـتبدال texCube2 و matCube2 بـــ texCube1 و matCube1.

الآن علينا أن نرسم القائمة ثنائيّة البعد:

```
' حفظ قيمة الخاصيّة في متغيّر احتياطي،
' لاستعادة قيمتها الأصليّة منه بعد رسم القائمة
Dim bPrev As Boolean = wireframe( )
' يجب تعطيل إمكانيّة رسم الأشكال كشبكة خطوط،
' لأنّنا لا نريد أن تبدو القائمة كذلك
wireframe = False
' تفعيل الشفافيّة، حتّى يتمّ استخدام لون الشفافية الذي اخترناه من قبل
' إنّ هذا سيزيل اللون القرمزيّ من خامة القائمة، ممّا سيمنحها حوافّ منحنية
D3DDev.RenderState.AlphaBlendEnable = True
' مصدر الرسم (القائمة)
D3DDev.SetStreamSource(0, vbMenu, 0)
' تنسيق الرسم
D3DDev.VertexFormat =
    CustomVertex.TransformedTextured.Format
' رسم الخامة على القائمة
D3DDev.SetTexture(0, texMenu)
' طريقة الرسم (شريط مثلثات).. تذكّر أنّ المستطيل يتكوّن من مثلّثين
D3DDev.DrawPrimitives(PrimitiveType.TriangleStrip,
    0, 2)
```

```
' إيقاف فاعلية الشفافية بعد رسم القائمة
D3DDev.RenderState.AlphaBlendEnable = False
' لا تنس إعادة قيمة خاصيّة شبكة الخطوط لقيمتها الأصليّة
wireframe = bPrev
```

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

```
D3DDev.RenderState.FillMode = FillMode.Solid
```

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

```
D3DDev.RenderState.FillMode = FillMode.WireFrame
```

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

```
fntOut.DrawText( _
    "Frame Rate: " + iFrameRate.ToString + "fps",
    New Drawing.Rectangle(5, 5 + (2 * (iFontSize + 6)),
    0, 0), _
    DrawTextFormat.Left Or DrawTextFormat.Top,
    Drawing.Color.FromArgb(255, 200, 128, 64))
```

لاحظ أنّ عليك أن تحدّد مساحة المستطيل لكي يحتوي الخطّ.. لا تحاول أن تجعله أكبر ممّا ينبغي، لأنّ ذلك يهبط بكفاءة تطبيقك.. عامّة يمكن استخدام بعض وسائل GDI، لمعرفة حجم المستطيل الذي يحتوي الخط.

استخدام الفئة:

الآن صارت لدينا فئة جاهزة.. نريد إذن استخدامها.

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

ابدأ بتعريف متغيّر من الفئة على مستوى النموذج:

```
Private c3DEngine As CSampleGraphicsEngine
```

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

```
c3DEngine = New CSampleGraphicsEngine(Me)
```

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

```
Dim iW, iH, iD As Integer
If CSampleGraphicsEngine.isDisplayModeOkay(1024,
    768, 32) Then
      iW = 1024 : iH = 768 : iD = 32
ElseIf CSampleGraphicsEngine.isDisplayModeOkay(
    1024, 768, 16) Then
      iW = 1024 : iH = 768 : iD = 16
ElseIf CSampleGraphicsEngine.isDisplayModeOkay(
    640, 480, 32) Then
      iW = 640 : iH = 480 : iD = 32
ElseIf CSampleGraphicsEngine.isDisplayModeOkay(
    640, 480, 16) Then
      iW = 640 : iH = 480 : iD = 16
```

```
Else
        ' لا يوجد طور عرض متاح!!
        Throw New Exception("No display modes found")
End If
c3DEngine = New CSampleGraphicsEngine(Me, iW,
        iH, iD)
```

بعد ذلك يمكننا ضبط خصائص الفئة:

```
c3DEngine.backFaceCulling = False
c3DEngine.drawFrameRate = True
c3DEngine.useTextures = True
c3DEngine.wireframe = False
bRunning = True
```

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

الآن كلّ ما علينا هو أن نكتب جملة تكراريّة يتمّ فيها تحديث الإطـــارات ورسمها.. سنعرّف أولا متغيّرا على مستوى النمـــوذج لنســـتخدمه فـــي الخروج من الجملة التكراريّة:

```
Dim bRunning As Boolean = True
```

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

```
Do While bRunning
        c3DEngine.oneFrameUpdate( )
        c3DEngine.oneFrameRender( )
        ' يجب استخدام الجملة التالية للســماح للنمــوذج بالاســتجابة
        لضغطات الأزرار
        Application.DoEvents( )
Loop
```

وأخيرا سنكتب حـــدث KeyDown، لنعطـــي فيـــه للمســـتخدم بعـــض الإمكانيّات:

```
Protected Overrides Sub OnKeyDown(ByVal e _
        As System.Windows.Forms.KeyEventArgs)
    Select Case e.KeyCode
        Case Keys.Escape ' إنهاء الحركة
            bRunning = False
        ' الانتقال من عرض الأجسام مغلقة إلى عرضها بالخطوط والعكس
        Case Keys.F1
            c3DEngine.wireframe =
                Not c3DEngine.wireframe
        ' استخدام الخامات أو منع استخدامها
        Case Keys.F2
            c3DEngine.useTextures =
                Not c3DEngine.useTextures
        ' محو الأوجه الخلفيّة أو تركها
        Case Keys.F3
            c3DEngine.backFaceCulling =
                Not c3DEngine.backFaceCulling
        ' كتابة معدّل رسم الإطارات أو عدم كتابته
        Case Keys.F4
            c3DEngine.drawFrameRate =
                Not c3DEngine.drawFrameRate
    End Select
End Sub
```

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

ما الذي تعلّمناه في هذا المشروع:

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

١- أهمّ مفاهيم العالم ثلاثيّ الأبعاد.

٢- تكوين المشهد باستخدام الرءوس Vertices والمثلّثات.

٣- كيف يعمل Direct3D9 بالتوافق مع نظامك.

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

٥- تحميل واستخدام الخامات.

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

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

كانت هذه خطوة صغيرة على بداية الدرب.

والله وليّ التوفيق.