

www.startimes2.com

دروس في الـ OpenGL

الدرس 1 : مقدمة + برنامجك الأول

khatibe_30@hotmail.fr



2010

بعد الكثير من التفكير, هل أبدأ كتابة دروس OpenGL أم لا؟ هل أستطيع إكمالها أم لا؟ هل سأكون مواظبا على كتابة الدروس بشكل منتظم أم لا؟ هل...الخ, قررت البدء في كتابة أول درس بإذن الله وباقى الدروس ستأتى إن شاء الله, فكتابة درس واحد أفضل من لا شيء.

قبل البدء في الدرس يجب أن أوضح بعض الأشياء:

أولاً:

هذه الدروس التي سأقدمها (إن شاء الله ستكون دروس) قد تحتوي على أخطاء أو تعليقات ضعيفة و لذلك يجب أن لا تعتبر أفضل مصدر لتعلم الـ OpenGL, ما ستفعله بالكود المرفق راجع لك و أنت تملك كل الحرية لاستعمال الكود كيفما تشاء و أينما تشاء, فأنا أحاول جعل عملية تعلم الـ OpenGL سهلة على المبتدئين في هذا المجال, أما إذا كنت جادا في تعلمه و تريد الإحتراف فعليك قراءة بعض الكتب المحترفة مثل :

🚩 Red Book Version 1.4

🚩 Blue Book Version 1.4

🚩 Orange Book Version 1.4

ولكن هي صعبة للفهم بالنسبة للمبتدئين, أيضا يجب أن تكون على إطلاع متوسط على لغة السي بلس بلس و التي سنستعملها في كتابة الأكواد.

ثانياً:

أثناء الدرس, قد أتجاوز شرح بعض الدوال عمدا و هذا لأنني في نفسي لم أفهمها بعد, و لذلك قد نأخذ قطعة كود و نفهمها جملة بدون التطرق إلى الدوال كل على حدا... قد تقول أنني مخطئ في هذا, و لكن فلسفتي في التعلم تقول أنه حين لا أفهم شيئا أظاهر أنني فهمته و أنتقل إلى الشيء الذي يليه و بهذا أحقق تقدما سريعا في التعلم, أما إذا قلت أنك لن تتقدم إلى الدالة الموالية إلا إذا فهمت الدالة الحالية فإني أضمن لك أنك لن تحقق تقدما أو على الأقل سيكون تقدمك بطيئا في تعلم أي شيء.

إذا لنبدأ على بركة الله...

1. مقدمة:

ما هو الـ OpenGL بالظبط؟ و من برمجته؟ و متى؟ و كيف هي بنيته الداخلية بالضبط؟ و...؟...الخ. لا يهمننا من برمجته و متى وكيف هي... و... و... كل ما نريد معرفته حاليا هو أن مكتبة الـ OpenGL و كل المكتبات المساعدة لها هي مكتبات عادية كغيرها من المكتبات تساعدنا على إنشاء عوالم ثلاثية الأبعاد و ثنائية الأبعاد بكفاءة عالية جدا, وكأنا نستعمل المكتبة stdio.h لعمليات الإدخال و الإخراج.

سيكون تقدمنا بطيئا في الدرس الأول و هذا لأنه الدرس الأول, في باقى الدروس سأكون سريعا جدا و أختصر الكثير لنحقق أكبر تقدم في أقل وقت, لذلك يجب عليك التركيز و أن تقرأ بجدية.

2. الأدوات اللازمة :

حسنًا... إلى هنا نحن نتفق على كل شيء، ما سنحتاجه هنا هو تثبيت أي مترجم س يبلس بلس و أنا أفضل أن يكون أحد نسخ الـ Visual C++ 2005 ، أنا سأستعمل الـ Visual C++ 2005 و لكن يمكنكم إستعمال أي نسخة و حتى استعمال أي مترجم غير مترجمات Microsoft .

بعد ذلك حمل مكتبة الـ OpenGL من هذا الرابط:

أنسخ المجلد GL إلى

C:\program files\Microsoft Visual Studio 8\VC\include\

أنسخ محتويات المجلد LIB إلى

C:\program files\Microsoft Visual Studio 8\VC\lib\

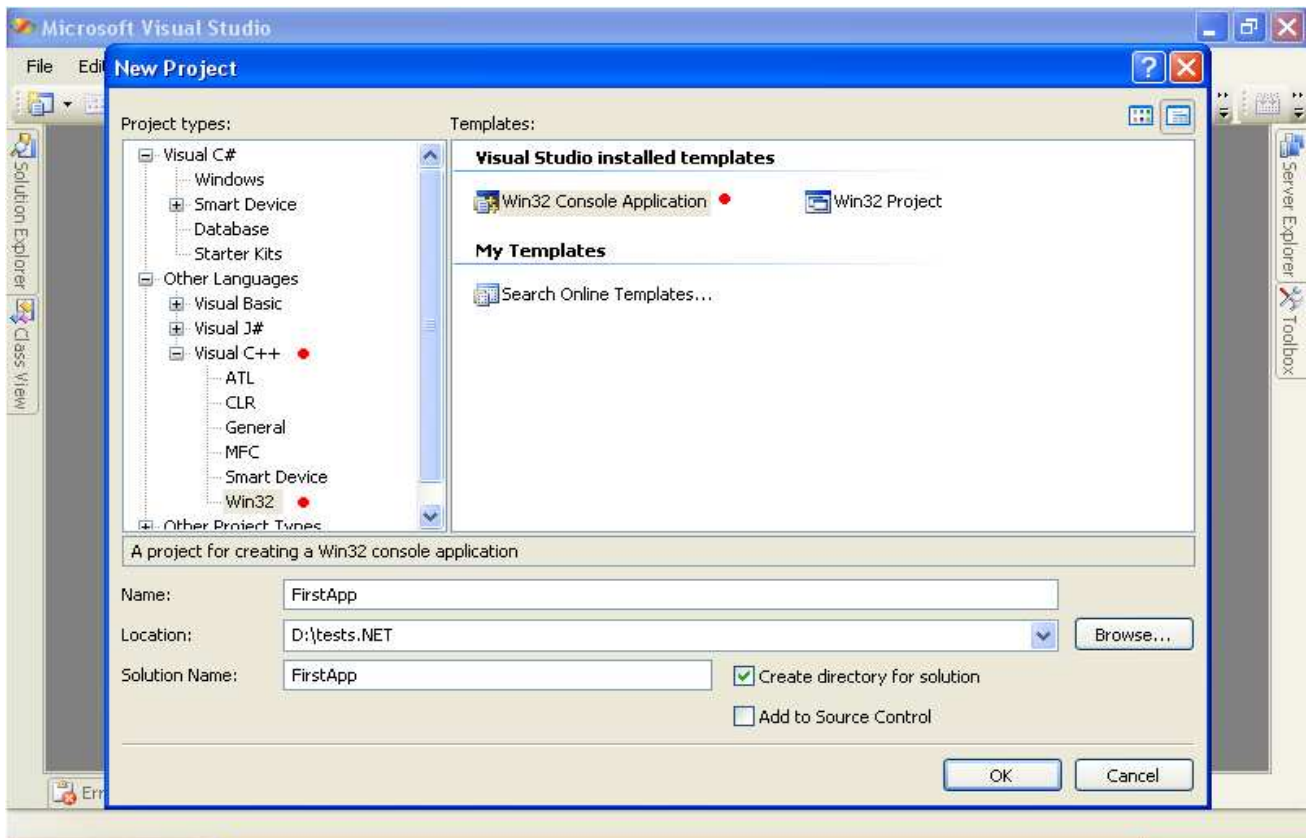
بعد ذلك قم بنسخ الملف المسمى glut32.dll إلى C:\WINDOWS\....

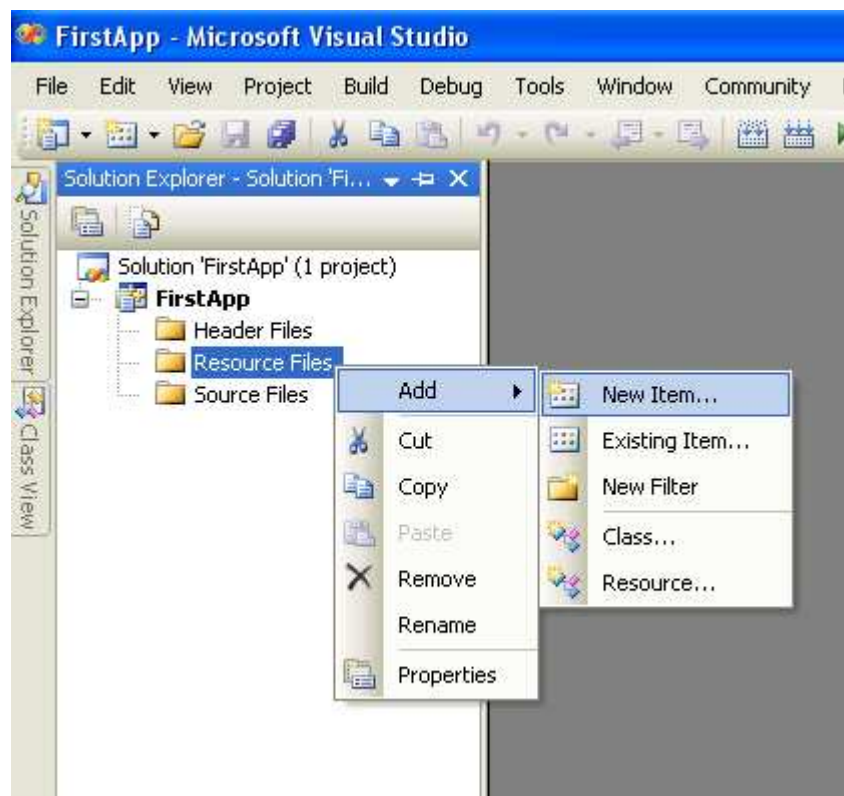
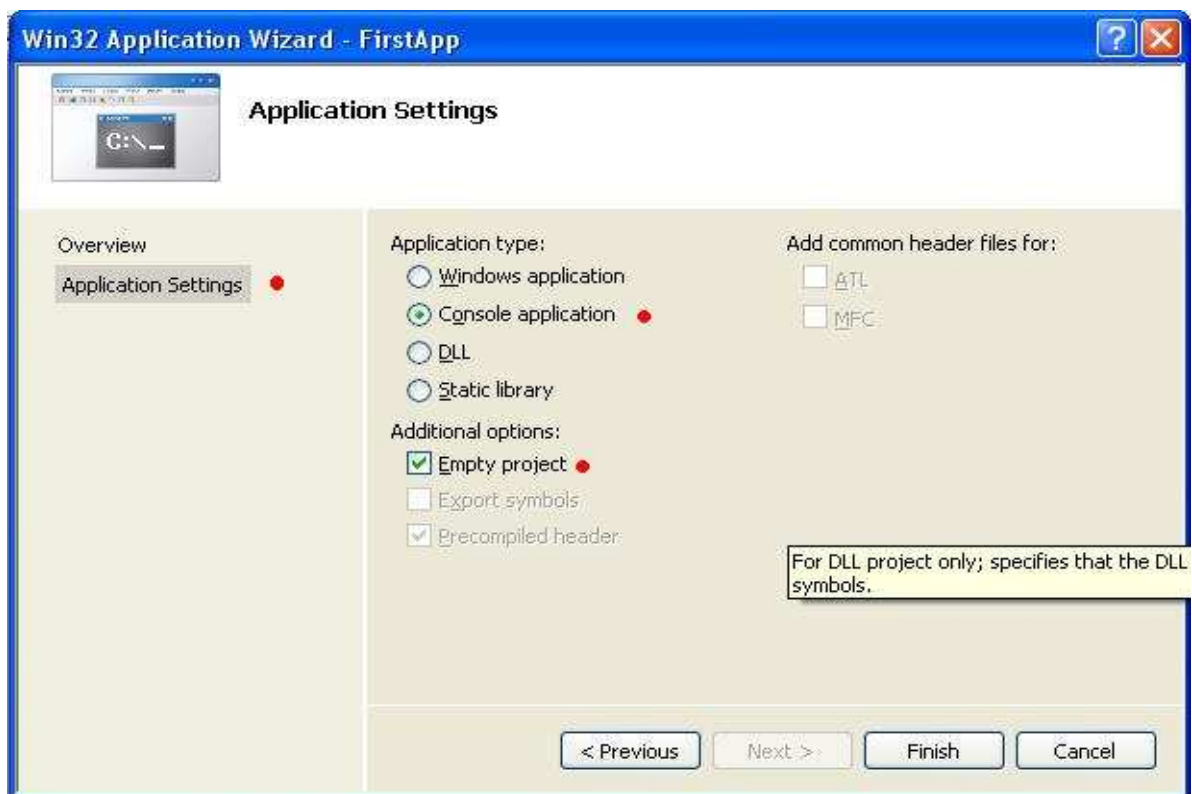
الآن نستطيع البدء...

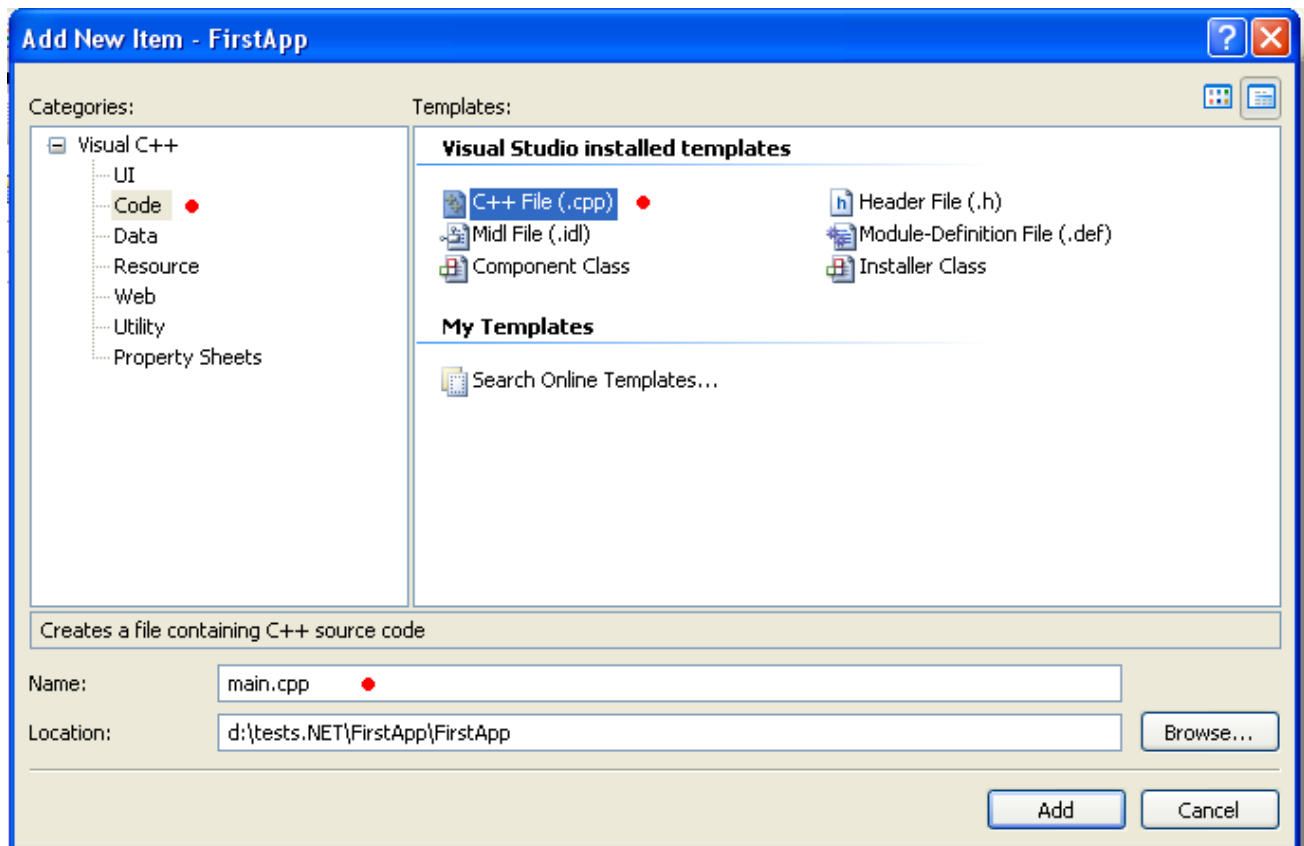
2. إنشاء أول نافذة:

أول ما يجب معرفته: كيف ننشئ نافذة OpenGL ؟

إفتح الفيجوال ستيديو و أنشئ مشروع جديد من نوع Console وسمه FirstApp كما في الصور:







طبعا الصور كانت معبرة جدا و لن أعقب عليها اختصارا للوقت, هنا نبدأ بكتابة بعض الأكواد في الملف `main.cpp`, قبل البدء ستلاحظ أن دوال OpenGL تكون من الشكل :

`glVertex3f(...)` ;

بداية الدالة هي **gl** وهذا يعني أن هذه الدالة موجودة في الملف `gl.h`, وقد تكون البداية `glu` وهذا يعني أنها موجودة في `glu.h`, وإذا كانت `aux` فإن الدالة موجودة في الملف الرأسي `glaux.h` وإذا كانت `glut` فالدالة تنتمي إلى `glut.h`.

المقطع الثاني **Vertex** هو إسم الدالة.

المقطع الثالث **3** يدل على عدد بارامترات الدالة.

أما بالنسبة للمقطع الرابع و هو **f** فهو يدل على نوع البارامترات, في هذه الحالة **f** يعني `float` و قد يكون **i** يعني `int`. بعد هذه المقدمة البسيط عن معنى إسم الدوال نكتب أول برنامج لنا و سنقوم برسم شكل هندسي ثنائي الأبعاد, سننتقل إلى العوالم الثنائية الأبعاد قريبا فلا تكن متسرعا...

أول شيء هو ربط البرنامج بالمكتبات اللازمة :

```
#include <stdlib.h>
#include <windows.h>
#include <GL/glut.h>
```

بما أننا سنربط الـ OpenGL بمشروع من نوع Console فإننا سنستعمل الدالة الرأسية glut.h ولهذا يجب عليك إرفاق الملف glut32.dll مع تطبيقاتك إذا أردتها أني تنفذ على حواسيب أخرى، ولكن وفي دروس أخرى إن شاء الله سنربط الـ OpenGL مع مشاريع Win32 و MFC وبذلك لن نحتاج إلى هذه المكتبة.

كل ما نريد رسمه نكتبه في دالة واحدة ونسميها مثلا Display و ليكن الكود المبدئي لها كالاتي:

```
void Display(void)
{
    /* نقوم بمسح النافذة باللون الأسود */
    glClearColor(0.0, 0.0, 0.0, 0.0);
    /* بعد ذلك نمسح الذاكرة المؤقتة بنفس اللون */
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    /* في دروس متغير نقوم بإزاحة و تدوير المبدأ، بهذه الدالة نرجع كل شيء كما كان */
    glLoadIdentity();
    //////////////////////////////////////
    /* نضيف الكود المسؤول عن الرسم هنا */
    //////////////////////////////////////
    /* بعد ذلك نقوم برسم كل ما رسم على الذاكرة المؤقتة على الشاشة */
    glutSwapBuffers();
}
```

الدالة السابقة و إلى حد الآن لا ترسم شيئاً ولكنها تهيئ بيئة الرسم.
الآن يجب أن ننشئ نافذة OpenGL و نعرضها و هذا يتم داخل الدالة الرئيسية main:

```
void main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("OpenGLtest");
    glutDisplayFunc(render);
    glutMainLoop();
}
```

عند تنفيذ البرنامج سيعرض لنا نافذة OpenGL بحجم 500x500 وهذا حسب الدالة `glutInitWindowSize` في الزاوية اليسرى العليا للشاشة و قد حددنا هذا ب `glutInitWindowPosition` تحمل عنوان `OpenGLTest`, بعد ذلك أخبرنا الـ GLUT أن الدالة المشؤولة عن الرسم أو التي سينفذها لرسم محتوى النافذة هي `Display` وهذا عن طريق دالة السطر 8 من مقطع الكود السابق.

هناك شيء آخر سنضيفه, عندما يتغير حجم النافذة يجب الأخذ بعين الاعتبار الحجم الجديد للنافذة, لذلك نضيف دالة جديدة إسمها `Reshape` وبالمرة سوف نقوم بتحديد بعض الأشياء فيها كمنفذ الرؤية و موقع الكاميرا:

```
void reshape(int w, int h)
{
    //w,h عرض النافذة الحالي
    glViewport(0, 0, w, h); /*تحديد منفذ الرؤية و هو يشمل كل النافذة*/
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (float)w/(float)h, 1.0, 800.0);
    gluLookAt(0,0,6,0,0,0,0,1,0);
    glMatrixMode(GL_MODELVIEW);
}
```

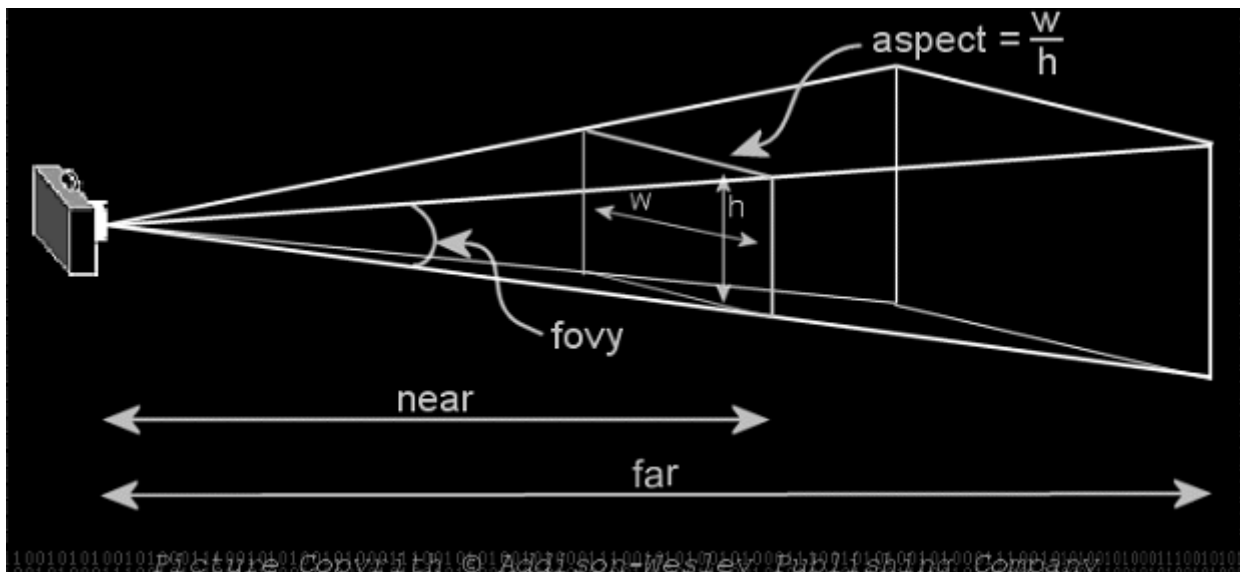
أهم دالة هي `gluPerspective`, ماهي؟ تستحق الشرح لأنها تحدد أبعاد الرؤية

مجسم الرؤية يمكن إنشاءه بنوعين من الإسقاط :

Perspective projection 

في هذا النوع, الأجسام المرسومة تتأثر بالمسافة, أي أنه كلما ابتعد الجسم صغر حجمه, ونستعمله عن طريق الدالة `gluPerspective(fovy, aspect, near, far);`

نوضح ماهية البارامترات عن طريق هذا الرسم:



Orthographic projection

في هذه الحالة, الأجسام المرسومة لا تتأثر بالبعد عن الكاميرا, مثل النصوص و القوائم, ونستعمل هذا الإسقاط عن طريق الدالة `glOrtho` و بما أننا لن نستعملها الآن فلا داعي لشرحها.

أما الدالة `gluLookAt` فنحدد بواسطتها موقع الكاميرا, وهي معرفة كالآتي:

```
gluLookAt (double eyex, double eyey, double eyez,  
double centerx, double centery, double centerz,  
double upx, double upy, double upz);
```

طبعا `eyex` و `eyey` و `eyez` يحددون موقع العين أو الكاميرا, `centerx` و `centery` و `centerz` يحددون النقطة التي تركز عليها الكاميرا, أما الباقي فسأخبركم عندما أتأكد من وظيفتهم بالظبط.

نسبنا شيئاً مهماً, و هو إخبار الـ `GLUT` أن ينفذ الدالة `Reshape` عند تغيير حجم النافذة, و لنفعل ذلك نضيف نضيف هذا السطر إلى الدالة `main` : `glutReshapeFunc(Reshape);`.

```

#include <windows.h>
#include <GL/glut.h>

void Reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (float)w/(float)h, 1.0, 800.0);
    gluLookAt(0,0,6,0,0,0,0,1,0);
    glMatrixMode(GL_MODELVIEW);
}

void Display(void)
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glutSwapBuffers();
}

void main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize( 500,500);
    glutInitWindowPosition (0,0);
    glutCreateWindow("OpenGLtest");
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutMainLoop();
}

```

أول برنامج مهم جدا مع أنه لا يرسم أي شيء, لذلك, سأكتب الكود كاملا للبرنامج :

حسنا... شغل البرنامج, ماذا يرسم؟ لاشيء !!!

من هنا نبدأ, سنرسم أول شكل لنا, أين نكتب الكود في رأيك ؟ اممم لم أسمعك... أحسنت, نتجه إلى الدالة Display ونقوم بكتابة قطع الكود المسؤولة عن الرسم.

سنرسم مثلث, كل هذا التعب من أجل مثلث ؟ أين العوالم الثلاثية الأبعاد؟ ... لا تستعجل, فقط الخطوات الأولى بطيئة, عندما نثبت أقدامنا و نفهم الأساسيات فبالكاد ستجاري هذه الدروس لسرعتها (إن شاء الله), لذلك يقال : الألف ميل تبدأ بخطوة.

سنحتاج إلى رسم ثلاث نقاط و توصيل خطوط بينها، نستعمل الدالة `glVertex3f(x, y, z)` لرسم النقاط، هذه الدالة

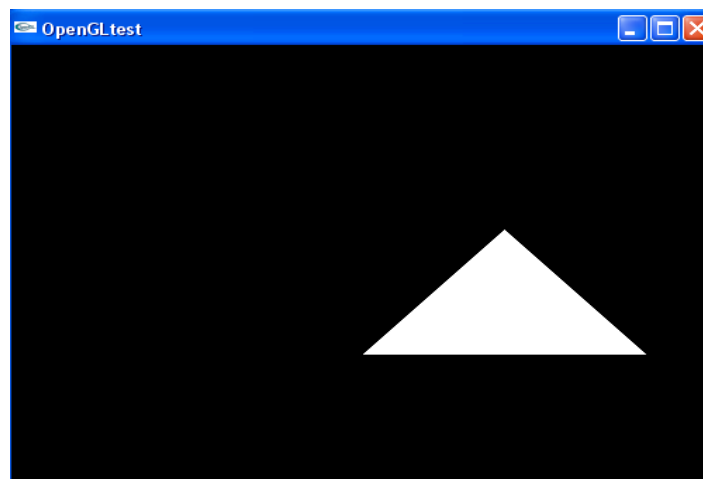
```
void Display(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glBegin(GL_TRIANGLES);
        glVertex3f(1.0, 1.0, 0.0);
        glVertex3f(0.0, 0.0, 0.0);
        glVertex3f(2.0, 0.0, 0.0);
    glEnd();
    glutSwapBuffers();
}
```

يجب أن تستدعى بين `glBegin` و `glEnd` :

الدالة `glBegin` تأخذ احدى البارامترات التالية:

رسم كل نقطة	GL_POINTS
ترسم الخط الواصل بين النقطة الأولى والثانية ثم الثالثة والرابعة وهكذا	GL_LINES
ترسم خط بين كل نقط والتي قبلها	GL_LINE_STRIP
ترسم كما في التي قبلها مع وصل النقطة الأخيرة بالأولى	GL_LINE_LOOP
كما في التي قبلها ولكن ترسم المضلع بدلا من حدوده	GL_LINE_POLYGON
كل أربع نقاط وكأنها مضلع لوحده مثلا 1 و2 و3 و4 مضلع ثم 5 و6 و7 و8 مضلع ثاني وكذا	GL_QUADS
كل ثلاث نقاط وكأنها مضلع لوحده	GL_TRIANGLES
النقاط الثلاث الأولى ترسم مضلع وكل نقطة جديدة مع آخر اثنتين قبلها تشكل المضلع الجديد	GL_TRIANGLE_STRIP

بعد تنفيذ البرنامج تكون النتيجة كما في الصورة :

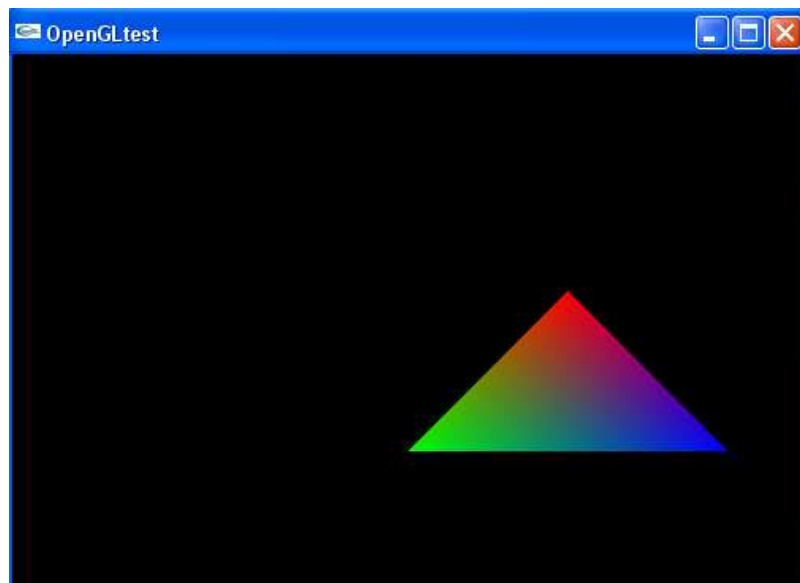


ما هذا المثلث الميت، لنضيف إليه بعض الألوان، باستعمال الدالة glColor4f(red, green, blue, alpha) نضفي بعض الألوان على مثلثنا الصغير :

```
void Display(void)
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glBegin(GL_TRIANGLES);
        glColor4f(1, 0, 0, 1);/*أحمر*/
        glVertex3f(1.0, 1.0, 0.0);
        glColor4f(0, 1, 0, 1);/*أخضر*/
        glVertex3f(0.0, 0.0, 0.0);
        glColor4f(0, 0, 1, 1);/*أزرق*/
        glVertex3f(2.0, 0.0, 0.0);
    glEnd();
    glutSwapBuffers();
}
```

تكون قيم الألوان محصورة في المجال [0, 1], أي أن 1 يمثل 255 وهي أعلى قيمة.

الآن, تكون النتيجة كالآتي:



يمكننا تغيير موقع الكاميرا, مثلا إذا أردنا أن نبعد الكاميرا لمسافة أكبر حتى يظهر الشكل أصغر فإننا نغير في بارمترات الدالة :

```
gluLookAt(0,0,10,0,0,0,0,1,0);
```

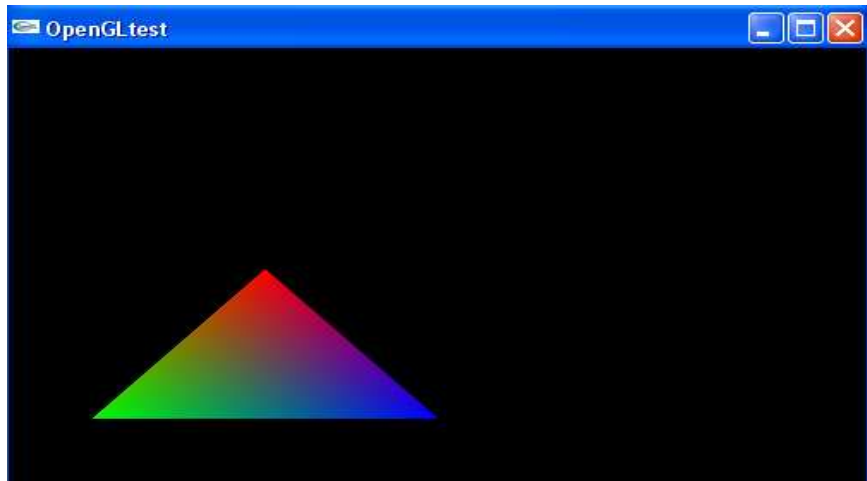
بعد هذا التغيير سيبدو المثلث أصغر.

3. الإزاحة و التدوير:

إذا أردنا تغيير مكان أو بالأحرى إزاحة المثلث فماذا نفعل ؟ هل نغير في إحداثيات النقاط المكونة للمثلث؟ هذا سيعقد الحسابات, لذلك نزيح مركز المعلم بأكمله, أي أننا نزيح المركز 0,0,0 ليصبح مثلا 0,1,0 و نرسم المثلث بنفس الإحداثيات, هكذا سنحرك الأجسام في المستقبل و لكن انتبه... إذا أزحنا أو أدركنا المركز يجب علينا إستدعاء الدالة glLoadIdentity() لإرجاع الأمور كما كانت... و هذا مثال :

```
void Display(void)
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(-2.0, 0.0, 0.0);
    glBegin(GL_TRIANGLES);
        glColor4f(1, 0, 0, 1);
        glVertex3f(1.0, 1.0, 0.0);
        glColor4f(0, 1, 0, 1);
        glVertex3f(0.0, 0.0, 0.0);
        glColor4f(0, 0, 1, 1);
        glVertex3f(2.0, 0.0, 0.0);
    glEnd();
    glutSwapBuffers();
}
```

والنتيجة تكون :



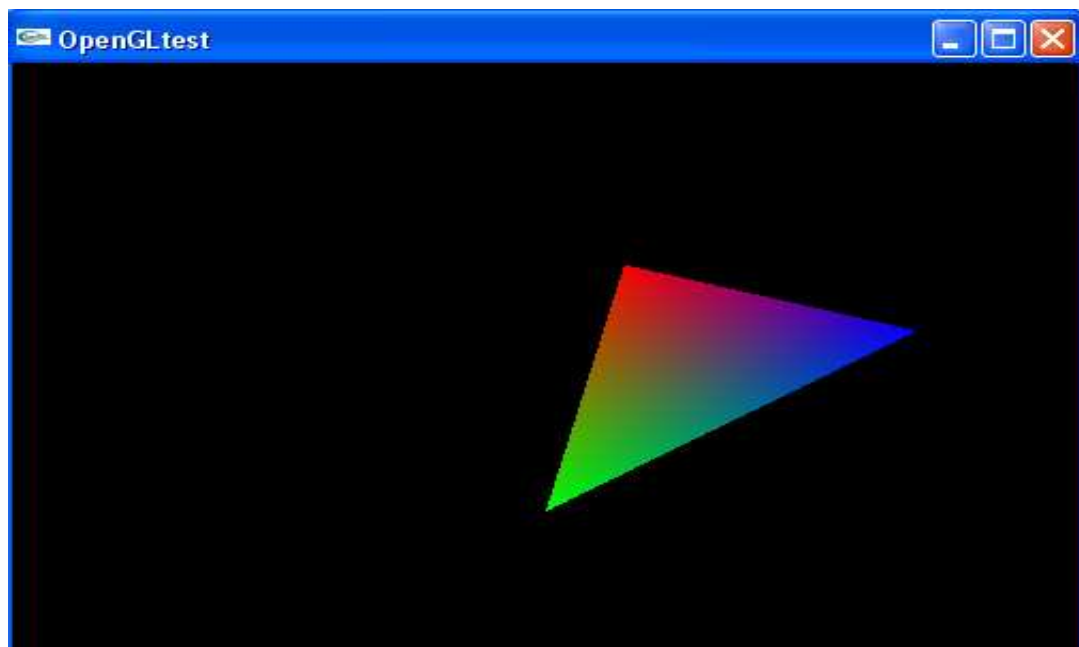
ماذا حدث للتو ؟ لقد أزحنا المركز x بقيمة -2, إذ أن الدالة glTranslatef(x, y, z) تقوم بإضافة x و y و z إلى الإحداثيات القديمة للمركز فتصبح إحداثيات المركز الجديد: (oldx + x, oldy + y, oldz + z) .

الآن جاء وقت تدوير المثلث, ماذا نفعل لندير المثلث بزاوية 45 درجة مثلا ؟ هل نجري معادلة رياضية معقدة للحصول

```
void Display(void)
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(30,0,0,1);
    glBegin(GL_TRIANGLES);
        glColor4f(1, 0, 0, 1);
        glVertex3f(1.0, 1.0, 0.0);
        glColor4f(0, 1, 0, 1);
        glVertex3f(0.0, 0.0, 0.0);
        glColor4f(0, 0, 1, 1);
        glVertex3f(2.0, 0.0, 0.0);
    glEnd();
    glutSwapBuffers();
}
```

على الإحداثيات الجديدة؟ بالطبع لا... فقط سنقوم بتدوير محاور المعلم الثلاثي الأبعاد :

الدالة `glRotatef(angle, x, y, z)` تدور الجسم بدرجة `angle` على المحاور الثلاثة أو إحداها, فمثلا في الكود السابق أدرنا المعلم بـ 30 درجة حول المحور `z` لتكون النتيجة:



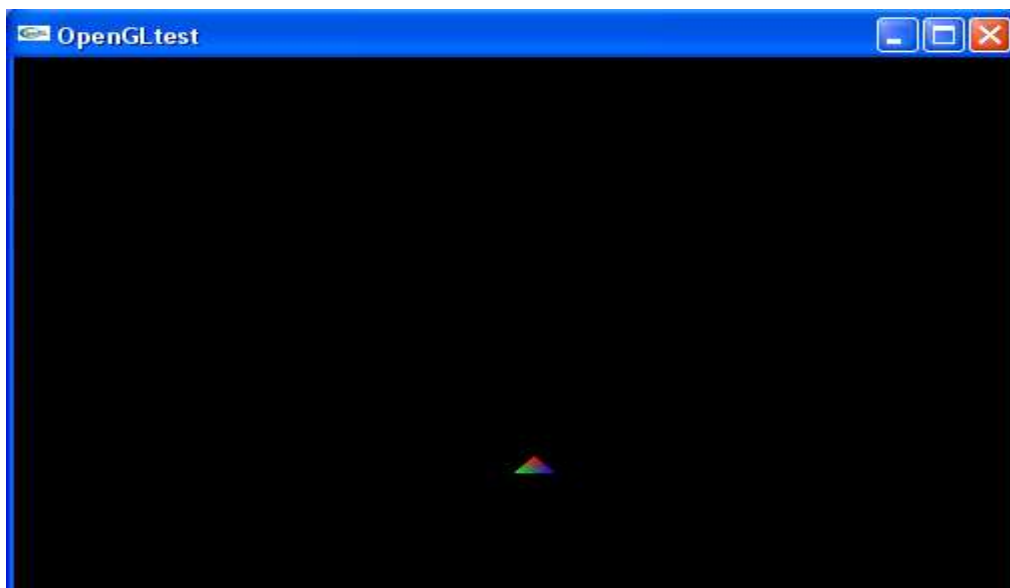
الدالة الأخيرة قبل أن نخرج من هذا الموضوع هي $glScalef(x, y, z)$ والتي تقوم بتحجيم الوحدة في المعلم، أشياء رياضية بسيطة، فيها يمكننا تكبير و تصغير الأجسام و الوحدة المستعملة للرسم، باستعمال هذه الدالة سنقوم بتصغير الوحدة المستخدمة للرسم، فكما تلاحظون فإن النقاط الثلاث المستعملة لرسم المثلث هي $(2,0)$; $(0,0)$, $(1,1)$ و إذا أردنا تصغير المثلث سنستعمل إحداثيات بالفاصلة كـ 0.2 و غيرها.

لتجنب ذلك نصغر الوحدة المستخدمة للرسم كي نتجنب الأعداد الصغيرة و التي تخط الحسابات، نغير الدالة Display إلى هذا الكود:

```
void Display(void)
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glScalef(0.1,0.1,0.1);
    glBegin(GL_TRIANGLES);
        glColor4f(1, 0, 0, 1);
        glVertex3f(10.0, 10.0, 0.0);
        glColor4f(0, 1, 0, 1);
        glVertex3f(0.0, 0.0, 0.0);
        glColor4f(0, 0, 1, 1);
        glVertex3f(20.0, 0.0, 0.0);
    glEnd();
    glutSwapBuffers();
}
```

بعد التنفيذ لن تتغير النتيجة – قمنا بالاستغناء عن الإزاحة و التدوير لعدم حاجتنا لذلك الآن – ولكن كما ترون أصبحنا نستعمل إحداثيات كبيرة لرسم نقاط المثلث $(20,0)$ $(0,0)$ $(10,10)$ وهذا أسهل للفهم...

الآن ننقص من حجم المثلث و هذا بتغيير إحداثياته إلى $(2,0)$ $(0,0)$ $(1,0)$ كما كان فتكون النتيجة:



حسنًا، نحن نحقق تقدماً جيداً هنا، الخطوة التالية هي أن نقوم بتحريك المثلث بواسطة الأسهم في لوحة المفاتيح، أولاً وكما قلنا للـ GLUT أن ينفذ الدالة Reshape عند تغيير حجم النافذة، علينا أن نقول له: نفذ الدالة Key() عند الضغط على أحد أزرار لوحة المفاتيح، بعد ذلك نحرك المثلث بحسب لوحة المفاتيح.

نضيف الدالة Keys إلى الكود و تكون كالآتي:

```
void Key(unsigned char key, int x, int y )
{
    if(key == 27 )
        exit(0);
}
```

ماذا تفعل هذه الدالة ؟ لا شيء سوى أنها تغلق البرنامج في حال وجدت أن الزر المضغوط على لوحة المفاتيح هو ESC.

حسنًا، نضيف سطر إلى الدالة main لنخبر الـ GLUT عن دالتنا Key الجديدة:

```
void main(int argc, char **argv)
{
    //...
    /* تجنباً لإضاعة المساحة سأكتب فقط السطر الذي قبل و بعد السطر الجديد... */
    /* الباقي لا يتغير */
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Key);
    // هذه الدالة تقوم بملأ الشاشة إذا أرد استعمالها
    glutFullScreen();
    glutMainLoop();
}
```

مهلاً مهلاً... لا يمكننا الاستفادة من أزرار الأسهم وكذلك F1, F2,...F12 لأنها أزرار خاصة و تحتاج إلى دالة خاصة، سنسمي هذه الدالة الخاصة SpecialKeys:

```
void SpecialKeys(int key, int x, int y)
{
    switch (key)
    {
        case GLUT_KEY_UP: /* تحرك المثلث إلى الأعلى */ break;
        case GLUT_KEY_DOWN: /* تحرك المثلث إلى الأسفل */ break;
        case GLUT_KEY_LEFT: /* تحرك المثلث إلى اليسار */ break;
        case GLUT_KEY_RIGHT: /* تحرك المثلث إلى اليمين */ break;
    }
}
```

```
void main(int argc, char **argv)
{
    //...
    glutKeyboardFunc(Key);
    glutSpecialFunc(SpecialKeys);
    glutMainLoop();
}
```

وكالعادة، نخبر الـ GLUT عن هذه الدالة:

جيد جداً... الآن و بعد تهيئة الجو —كما نقول— نكتب الكود الذي سيغير موقع المثلث، كيف هذا؟؟ سؤال محير؟؟ طبعاً لا.. سنحرك المثلث على المحاورين x و y فقط، لذلك نضيف متغيرين movex و movey حيث نضع فيهما القيمة التي يجب أن نضيفها للإحداثيات القديمة، و هذه القيمة تكون موجبة إذا ضغطنا على فوق أو يمين في لوحة المفاتيح، و قد تكون سالبة وهذا إذا ضغطنا على المفتاح يسار أو تحت من لوحة المفاتيح... سهل جداً أليس كذلك؟

وبما أننا تعلمنا أن نحرك المعلم بأكمله بدل تغيير الإحداثيات كل على حدا فإننا سنستعمل الدالة glTranslatef .

لنبدأ، نصرح في أعلى الملف عن المتغيرين الجميلين و هما:

```
float movex = 0,
      movey = 0;
```

مباشرة إلى الدالة Display لإضافة سطر جديد كي تصبح الدالة كالآتي:

```
void Display(void)
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glScalef(0.1,0.1,0.1);
    glTranslatef(movex, movey, 0);
    glBegin(GL_TRIANGLES);
        glColor4f(1, 0, 0, 1);
        glVertex3f(1.0, 1.0, 0.0);
        glColor4f(0, 1, 0, 1);
        glVertex3f(0.0, 0.0, 0.0);
        glColor4f(0, 0, 1, 1);
        glVertex3f(2.0, 0.0, 0.0);
    glEnd();
    glutSwapBuffers();
}
```

الآن بقي لنا أن نحدث قيم كل من movey و movey في كل مرة يتم فيها الضغط على المفاتيح و بعدها نحدث الرسم, و نفعل ذلك بإضافة بعض الأوامر إلى الدالة SpecialKeys :

```
void SpecialKeys(int key, int x, int y)
{
    switch (key)
    {
        case GLUT_KEY_UP: movey++; Display(); break;
        case GLUT_KEY_DOWN: movey--; Display(); break;
        case GLUT_KEY_LEFT: movex--; Display(); break;
        case GLUT_KEY_RIGHT: movex++; Display(); break;
    }
}
```

واضح جدا ما فعلناه, حدثنا ثم رسمنا... لا شيء يستحق الشرح, الـ 'ن' قم بتنفيذ البرنامج و قل لي ما العيب الذي به ؟ ألا يوجد هناك نقص ؟ أكيد يوجد.... ما هو؟ نلاحظ أنه عند الضغط على المفتاح فوق مثلا في زر المفاتيح يتحرك المثلث إلى الأعلى ثم يتوقف ثم يواصل التحرك, لا يجب أن يكون الأمر هكذا, ماذا نفعل لتفادي المشكل؟؟؟

لا يوجد حل؟ أكيد هناك حل... سنضيف دالة جديدة لنستدعيها عند رفع الإصبع من على المفتاح.... ففهمتم الفكرة إذا... سننشئ متغير جديد بحيث إذا ضغطنا على المفتاح فوق يحمل ذلك المتغير القيمة 1 و نبدأ بتحريك المثلث و لا نتوقف إلا حين نرفع الإصبع عن المفتاح و الدالة الجديدة SpecialKeysUP تكون مسؤولة عن هذه العملية, أي عملية رفع الإصبع عن المفتاح.

أولا نضيف مصفوفة من 255 عنصرا تحمل بيانات المفاتيح, 1 تعني أن المفتاح مضغوط, 0 يعني أن المفتاح غير مضغوط:

```
float movex = 0,
      movey = 0;
bool keys[255];
```

نغير الدالة السابقة SpecialKeys كليا لتصبح كالآتي :

```
void SpecialKeys(int key, int x, int y)
{
    keys[key] = true;
}
```

ما فعلناه هو أننا سنعطي القيمة true للمفتاح المضغوط، من هنا يجب علينا إضافة دالة أخرى وهي SpecialKeysUP لكي تقوم بإعطاء القيمة false للمفتاح حين ينتهي الضغط عليه :

```
void SpecialKeysUP(int key, int x, int y)
{
    keys[key] = false;
}
```

وكل مرة يجب أن نخبر الـ GLUT أننا أضفنا دالة أخرى لمعالجة حدث رفع الضغط عن المفاتيح، إذا نضيف سطرًا في الدالة الرئيسية main لتصبح:

```
void main(int argc, char **argv)
{
    //...
    glutSpecialFunc(SpecialKeys);
    glutSpecialUpFunc(SpecialKeysUP);
    glutMainLoop();
}
```

بعد هذا، يلزمنا إضافة دالة جديدة للحركة و تقوم بتحديث النافذة باستمرار وهذا باستدعاء Display وكأنها حلقة غير منتهية، طبعًا لن تكون لها فائدة إلا إذا أخبرنا الـ GLUT أنها هي دالة الحركة، يقوم GLUT بتنفيذ هذه الدالة مادامت النافذة لا تستقبل أحداث جديدة، وهذا يفيد بالعرض، يمكننا طبعًا إستعمال Timer ولكن حاليًا سنستعملها هي، نسمي هذه الدالة action وهي كما يلي :

```
void action(void)
{
    if(keys[GLUT_KEY_UP])
        movey++;
    if(keys[GLUT_KEY_DOWN])
        movey--;
    if(keys[GLUT_KEY_LEFT])
        movex--;
    if(keys[GLUT_KEY_RIGHT])
        movex++;
    Display();
}
```

الكود المكتوب بها واضح جدًا، فهي تتحقق من القيم الموجودة في كل متغير خاص بمفاتيح الأسهم الأربعة و تقوم بزيادة أو إنقاص أحد المتغيرين movex أو movey بحسب المفتاح المضغوط ثم ترسم النافذة من جديد.

نضيف هذا السطر إلى الدالة main لتكون الدالة action فعالة:

```
void main(int argc, char **argv)
{
    //...
    glutIdleFunc(action);
    glutMainLoop();
}
```

الآن نفذ البرنامج و سترى أننا حصلنا على نتيجة مقبولة جدا مبدئيا.

أصبح المثلث الصغير يتحرك بسلاسة كبيرة, هل أنهينا الدرس الأول ؟ لا... قبل أن نختم الدرس الأول نجعل المثلث يتحرك حركة دائرية متواصلة حول نفسه و هذا لإضفاء بعض الحركة على البرنامج الأول... كيف نفعل ذلك؟؟ بسيطة, هل تذكر الدالة glRotatef والتي قلنا أنها تدير المحاور بدرجة معينة ؟ جاء وقت الإستفادة منها.

نضيف إلى البرنامج متغير جديد نسميه angle ليحمل قيمة الزاوية التي سندور بها المعلم قبل رسم المثلث, ثم بعد ذلك نستخدم الدالة glRotatef للتدوير وانتهى, ليصبح الكود النهائي للبرنامج كالاتي:

```
#include <stdlib.h>
#include <windows.h>
#include <GL/glut.h>
#include <stdio.h>

void Reshape(int w, int h);
void Key(unsigned char key, int x, int y );
void SpecialKeys(int key, int x, int y);
void Display(void);

float movex = 0,
       movey = 0;
bool keys[255];
float angle = 0;

void Reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (float)w/(float)h, 1.0, 100.0);
    gluLookAt(0,0,6,0,0,0,0,1,0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void Key(unsigned char key, int x, int y )
{
    if(key == 27 )
        exit(0);
}
```

```
void SpecialKeys(int key, int x, int y)
{
    keys[key] = true;
}
void SpecialKeysUP(int key, int x, int y)
{
    keys[key] = false;
}
void action(void)
{
    if(keys[GLUT_KEY_UP])
        movey++;
    if(keys[GLUT_KEY_DOWN])
        movey--;
    if(keys[GLUT_KEY_LEFT])
        movex--;
    if(keys[GLUT_KEY_RIGHT])
        movex++;
    Display();
}
void Display(void)
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glScalef(0.1,0.1,0.1);
    angle = (int)(angle + 10) % 360;
    glTranslatef(movex,movey,0);
    glRotatef(angle,0,0,1);
    glBegin(GL_TRIANGLES);
        glColor4f(1, 0, 0, 1);
        glVertex3f(1.0, 1.0, 0.0);
        glColor4f(0, 1, 0, 1);
        glVertex3f(0.0, 0.0, 0.0);
        glColor4f(0, 0, 1, 1);
        glVertex3f(2.0, 0.0, 0.0);
    glEnd();
    glutSwapBuffers();
}
void main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize( 500,500);
    glutInitWindowPosition (0,0);
    glutCreateWindow("OpenGLtest");
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Key);
    glutSpecialFunc(SpecialKeys);
    glutSpecialUpFunc(SpecialKeysUP);
    glutIdleFunc(action);
    glutMainLoop();
}
```

إلى هنا نكتفي كدرس أول و إن شاء الله تعقبه دروس أخرى, قبل أن تغلق الدرس... ما رأيك أن نبرمج كعمل تطبيقي لهذا الدرس لعبة SNAKE المشهورة ؟ سيكون تطبيقا جيدا... ولكن نضرا لضيق الوقت سأذهب, و بالاعتماد على مهاراتك السابقة في البرمجة و على ما اكتسبته من هذا الدرس في عالم الرسومات أقترح أن تبرمجها بنفسك و ترسلها لي عن طريق الـ E-MAIL أو ضعها على شكل موضوع إذا كنت من أعضاء المنتدى و سأخذها من هناك , سأكون ممتنا لمساعدتك لي.