مقدمة في
OpenGL

إعداد : المهندسة دانيا

سوريا-حمص-٢٠٠٥

الفصل الأول

تعريف بمكتبة الـــ OpenGL

مقدمة:

لقد شكلت هذه المكتبة قاعدة ضخمة في مجال الرسوميات منذ وقت ليس بالقصير، وتعتبر حتى الآن النـــواة الأساسية لبناء مكتبات رسومية أكثر غنىً وتنوعاً، كما تؤمن هذه المكتبة الكثير من التوابع الرسومية التـــي تمكن المبرمج من بناء أي تصميم رسومي خاص بتطبيقه البرمجي، سواء كان لعبة تحتاج إلى إمكانيات بيانية ضخمة، أو برامج هندسية تحتاج إلى توابع خاصة وهامة. كما أنها مصممة بحيث يمكنها أن تعمل ضمن أي عتاد و في أي نظام عرض بياني لذلك فهي تمتلك العديد من الخصائص التي جعلتها من أهم وأقوى المكتبات البيانية في العالم .

تساعد هذه المكتبة على إنشاء صور ذات نماذج هندسية بسيطة، كما تزود مستخدميها بتوابع غاية في الأهمية للتحكم بالألوان والإضاءة والظلال والضبابية ، بالإضافة إلى إمكانية إخفاء السطوح (عند تراكب الأشـــكال) و إظهار الحواف الناعمة والخشنة في الصورة لإعطائها شكلاً يشبه شكلها الحقيقي ، وسندرس هذه المواضيع بشكل مفصل لاحقاً.

أما من الناحية الحركية، فهي تمتلك توابع خاصة تستخدم في التطبيقات ثلاثية الأبعاد لإعطاء هذه التطبيقات المظهر الحركي.

ولكن الـــ OpenGL غير قادرة على توفير الأغراض والكائنات الرسومية ثلاثية الأبعاد كالسيارات، جـــسم الإنسان، الجزيئات ...الخ ، وإنما يتم تمثيل الكائنات ثلاثية الأبعاد باستخدام خطوط منفصلة تتصل مع بعضها لإنشاء النموذج . كما أنها غير قادرة على فتح نوافذ في تطبيقاتها ولا إنشاء تطبيقات تتفاعل بمرونـــة مـــع المستخدم (Interactive Applications) بحيث تكون قادرة على الاستجابة للأحداث الخارجية . وقـــد تـــم التغلب على هذه المشاكل باستخدام المكتبة الخدميِّـــة [1](GLU) ومجموعـــة إجرائيـــات مـــسبقة التعريـــف [2](GLUT) .

- المكتبة الخدمية GLU : تستخدم دائماً مع مكتبة الـــ OpenGL وهي توفر ميـــزات النمذجـــة حيـــث تستخدم العناصر الأساسية للرسم – التي توفرها الـــ OpenGL – كالخطوط المـــستقيمة والمنحنيـــات ...الخ من الأشكال الهندسية البسيطة ومن ثم تنشئ علاقة بين هذه الأجسام في الفراغ ثنائي البعـــد أو ثلاثي البعد كما أنها توفر بعض الأشكال ثلاثية البعد .
- مجموعة إجرائيات مسبقة التعريف GLUT : وهي التي توفر إمكانية جعل التطبيقات الرسومية تفاعلية مع المستخدم من خلال تقديم إجرائيات تستجيب لأي حدث خارجي كأحداث لوحة المفاتيح (ضـــغط زر

---

[1] المكتبة الخدمية : The OpenGL Utility Library

[2] إجرائيات مسبقة التعريف : The OpenGL Utility Toolkit

مثلاً)، وأحداث الفأرة (Click,Double Click,Mouse over,..)، إضافة إلى توفير القدرة على إنشاء وفتح عدة نوافذ في هذه التطبيقات (Multi Forms). كما تزود مستخدميها بالعديد من الأشكال والكائنات ثلاثية الأبعاد دون الحاجة إلى تمثيلها باستخدام الخطوط المستقيمة العديدة وهي الطريقة التي كانت تستخدمها مكتبة الـ OpenGL، وهي طريقة معقدة وخاصة عندما يكون المشهد يحتوي العديد من الأشكال ثلاثية البعد وسنذكر على سبيل المثال بعض الأشكال التي يمكن استخدامها مباشرة كجسم كروي ،حلقة ، إبريق الشاي ، ...الخ .

هذه المكتبة مكتوبة بلغة C مما يعطي هذه المكتبة ميزة إضافية تحتاجها أي مكتبة رسومية وهي سرعة إنجاز الرسوم وتحريكها، حيث من المعروف أن لغة C من اللغات التي تتعامل مع لغة الآلة التي يفهمها أي معالج بطريقة سريعة للغاية ولذلك نحتاج عند التعامل معها إلى تضمين الملفات الخاصة بها وتسمى هذه الملفات بالملفات الرأسية أو ملفات الترويسة Header Files.

الملفات الرأسية الواجب تضمينها في التطبيق:

في التطبيقات الرسومية التي تستخدم الـ OpenGL يجب أن يضمَّن الملف الخاص بهذه المكتبة وهو gl.h، وبما أن المكتبتين GLU و GLUT تستخدمان دائماً مع المكتبة OpenGl فبالتالي يجب تضمين الملفات الرأسية لهما أيضاً و هما glu.h، glut.h على الترتيب.

مثال ١:

سنعرض الآن الهيكل العام للتطبيق الذي يستخدم مكتبة الـ OpenGL وهذا التطبيق يعرض شكل هندسي بسيط وهو مستطيل باللون الأبيض مع خلفية سوداء.

Example 1-1 : White Rectangle on a Black Background

```
#include <whateverYouNeed.h>

main() {

   InitializeAWindowPlease();

   glClearColor (0.0, 0.0, 0.0, 0.0);
   glClear (GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f (0.25, 0.25, 0.0);
      glVertex3f (0.75, 0.25, 0.0);
      glVertex3f (0.75, 0.75, 0.0);
      glVertex3f (0.25, 0.75, 0.0);
   glEnd();
   glFlush();

   UpdateTheWindowAndCheckForEvents();
}
```

ملف الترويسة

السطر الأول من هذا التطبيق هو الـــ **main()** وهي ترويسة الإجرائية الرئيسية ،والتي تتضمن استدعاء للإجرائية **InitializeAWindowPlease()** المسؤولة عن عملية تهيئة النافذة من إنشاء للنافذة الأساسية والنوافذ الجزئية و تحديد موقع النافذة على الشاشة وحجم وعنوان للنافذة وسيتم تفصيل كل من هذه التعليمات لاحقاً . أما التعليمات التي تلي استدعاء هذه الإجرائية فهي تعليمات الـــ OpenGl وهي :

١. **glClearColor()** : تحديد لون مسح النافذة وهنا تم تحديد اللون الأسود .
٢. **glClear()** : تنفيذ عملية المسح الفعلية ، وكلما تستدعى هذه التعليمة يتم مسح النافذة .
٣. **glColor3f()** : تحديد لون خط الرسم وفي هذا التطبيق تم اختيار اللون الأبيض .
٤. **glOrtho()** : تخصيص نظام إحداثيات مناسب يستخدم لرسم الشكل وبحسب نظـــام الإحداثيات تحدد إحداثيات الرسمة التي ستظهر على النافذة فمثلاً في المثال السابق تـــم تحديد نظام الإحداثيات ضمن المجال [٠,٠،٠,١]بالنسبة للـــX وبالمثل بالنسبة لـــY أمـــا بالنسبة للإحداثيين الأخيرين [−١,٠ ،١,٠] فهما خاصين بمجال الإحداثي Z.
٥. **glBegin()** ، **glEnd()** : يكتب ضمنهما تعليمات رسم الشكل المطلوب وفي هذا المثال تم رسم مضلع بأربع حواف .
٦. **glVertex3f()** : وهي تعليمة رسم نقطة بشكل عام وقد استخدمت هنـــا لتحديـــد الزوايا الأربعة للمضلع وتمثل البارمترات الممررة لهذه التعليمة الإحداثيات z،y،x مـــع ملاحظة أن قيم الـــ y،x ضمن المجال [0,1] للتناسب مع نظام الإحداثيات المستخدم والذي تم تحديده سابقاً ، في حين أن قيمة البارمتر الثالث والذي يمثل الإحداثي z قيمته = ٠ لأننا الآن لا نهتم بالبعد الثالث.
٧. **glFlush()** : وهي التعليمة الأخيرة المسؤولة عن إظهار الشكل ضمن النافذة بشكل فعلي لأن النقاط التي تم تعيينها سابقاً وضعت ضمن مخزن مؤقت [٣] ريثما يتم إظهارها دفعة واحدة على النافذة .

وأخيراً يتم استدعاء الإجرائية **UpdateTheWindowAndCheckForEvents()** وتتـــضمن العمليات المتعلقة بالأحداث والتي تجعل التطبيق البياني قادر على تلقي الأوامر من المستخدم .

ملاحظة:

هناك اجرائيتين هما :

InitializeAWindowPlease() ،UpdateTheWindowAndCheckForEvents()

---

3 مخزن مـؤقـت :buffer

هاتان الإجرائيتان ليستا من مكتبة الـــ OpenGl لأننا سبق وذكرنا أن الـــ OpenGl غير قادرة على إنشاء وتهيئة نوافذ في التطبيقات الرسومية، وكذلك ذكرنا أنها غير قادرة على جعل التطبيقات تفاعلية بحيث تتلاقى الأحداث من المستخدم، و بالتالي فإن التعليمات التي تتضمنها هاتين الإجرائيتين من مكتبة الـــ GLUT وهي المسؤولة عن إنجاز هذه العمليات كما ذكر آنفاً.

القواعد التي تحكم تعليمات مكتبة الـــ OpenGL:

من خلال المثال السابق نلاحظ ما يلي:

١- جميع تعليمات الـــ OpenGL تبدأ بالحرفين gl ثم يكتب الحرف الأول من التعليمة بحرف كبير.
٢- عندما تتألف التعليمة من أكثر من كلمة فإن كل كلمة تبدأ بحرف كبير أيضاً.
٣- أما الثوابت فتبدأ بالحرفين GL_ ويتبعها المحرف السفلي (_) ، وغالباً ما تتألف الثوابت من عدة مقاطع بحيث تكون جميع الأحرف كبيرة ويفصل بين كل المقاطع بالمحرف السفلي كما في الثابت GL_POLYGON الذي يمرر للتعليمة glBegin() لتحديد الشكل الذي سيتم رسمه و في مثالنا هو مضلع ، GL_COLOR_BUFFER_BIT وهو الثابت الذي يمرر للإجرائية glClear() .
٤-

كما يمكن ملاحظة وجود أحرف و أرقام إضافية على التعليمات مثل (**3f**) كما في التعليمة glColor3f()، glVertex3f() حيث من الواضح لدينا أنه في تعليمة تحديد اللون كان يكفي أن تكون التعليمة عبارة عن كلمة color، ولكن هناك عدة أشكال لهذه التعليمة. تتميز كل تعليمة عن الباقي بعدد الوسطاء الممررة لها ونوعها، وهو ما يظهر في اسم التعليمة (في مثالنا السابق ٣)،ففي مثالنا السابق يظهر الحرف f الذي يشير هنا إلى أن الوسطاء الممرة للتابع هي أرقام ذات فواصل عشرية (المحرف f اختصار لـــ float) .

تقبل الـــ OpenGL ثمانية أنماط لوسطائها (متحولات دخل التابع) والتي ستوضح في الجدول :

Suffix	**Data Type**	**Typical Corresponding C-Language Type**	**OpenGL Type Definition**
b	8-bit integer	signed char	Glbyte
s	16-bit integer	Short	Glshort
i	32-bit integer	int or long	GLint, Glsizei
f	32-bit floating-point	Float	GLfloat, Glclampf
d	64-bit floating-point	Double	GLdouble, Glclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, Glboolean

us	16-bit unsigned integer	unsigned short	Glushort
ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint, GLenum, Glbitfield

فمثلاً التعليمتين `glVertex2i(1, 3)`، `glVertex2f(1.0, 3.0)` لهما نفس العمل ولكن ما يختلفان به هو نوع الوسطاء الممررة فالأولى نوع وسطائها هو `integer`، أما الثانية فهو `float` . بالإضافة إلى أنه توجد تعليمات يمكن أن تنتهي باللاحقة `v` حيث يشير هذا المحرف أن الوسيط الممرر للتعليمة عبارة عن شعاع أو مصفوفة .

مثال:

```
glColor3f(1.0, 0.0, 0.0);
GLfloat color_array[] = {1.0, 0.0, 0.0};
glColor3fv(color_array);
```

إدارة النافذة:

تقوم مكتبة الـ `GLUT` بعمليات الإدارة التي تتعلق بالنوافذ من إنشاء وتهيئة الحجم والموقع وهناك خمسة تعليمات تستخدم بشكل أساسي لإتمام هذا العمل ، وهذه التعليمات هي :

١. **`glutInit(int *argc, char **argv)`** : تابع يستخدم لتهيئة مكتبة الـ `GLUT`، مع الأخذ بعين الاعتبار للوسطاء التي تؤخذ من محرر الأوامر [4].

٢. **`glutInitDisplayMode(unsigned int mode)`** : تابع لإعداد وضع الشاشة الابتدائي ودقة الـ `Buffer` المستخدم ، وتحديد نظام الألوان المستخدم هل `RGBA` أو `color-index` وقد تستدعى هذه التعليمة بالأسلوب التالي:

```
glutInitDisplayMode (GLUT_SINGLE| GLUT_RGB)
```

و تتغيّر هذه الثوابت بحسب طبيعة التطبيق الرسومي، فإذا كان التطبيق عبارة عن أشكال ساكنة وغير متحركة فإننا نستخدم الثابت `GLUT_SINGLE` ولكن عندما تكون الأشكال متحركة عندها يتوجب استخدام الثابت `GLUT_DOUBLE` .

---

[4] محرر الأوامر: `Command Line`

٣. **glutInitWindowPosition(int x, int y)** : وهو تابع مسؤول عن تحديد مكان ظهور النافذة أمام المستخدم عند تشغيل التطبيق .

٤. **glutInitWindowSize(int width, int size)** :وهو تابع مسؤول عن تحديد حجم نافذة التطبيق .

٥. **int glutCreateWindow(char *string)** :وهو تابع مسؤول عن إنشاء النافذة التي سيظهر ضمنها الأشكال الرسومية وتعيد رقم للنافذة التي تم إنشاؤها والذي سنستفيد منه لاحقاً ، أما الوسيط الممر لها فهو عبارة عن العنوان الذي سيظهر في رأس النافذة .

٦. **glutMainLoop()** :وهو تابع للدخول في حلقة معالجة الأحداث في GLUT، بعد الانتهاء من التهيئة .

إن الإجرائية **InitializeAWindowPlease()** التي تم ذكرها سابقاً هي التي تضم تعليمات إدارة النافذة ، أما فيما يتعلق بتعليمات الإجرائية **UpdateTheWindowAndCheckForEvents()** فهي التعليمات المسؤولة عن الاستجابة للأحداث التي تصدر عن المستخدم وهذه الأحداث مصدرها الفأرة أو لوحة المفاتيح ...الخ . وسيتم شرح تعليمات هذه الإجرائية في الفصل القادم .

إجرائية العرض وإظهار الأشكال:

ذكرنا أن المثال السابق يقوم بإظهار مضلع، وقد رأينا أن تعليمات رسم المضلع توضع ضمن تعليمتي **glBegin()** و**glEnd()** ولكن المكان الذي تم وضع التعليمات فيه غير صحيح ،حيث تم وضع كتلة التعليمات هذه مباشرة في إجرائية الـ main الرئيسية ، ولكن تخيل أنك تريد رسم أكثر من شكل وأكثر من مشهد فمن غير المنطقي وضع كل تعليمات الرسم مباشرة ضمن الإجرائية الرئيسية، وإنما يجب وضعهم ضمن إجرائية خاصة بالعرض بحيث يتم استدعاؤها لاحقاً. ولكن ستختلف هنا طريقة الاستدعاء عما كنت قد تعلمته في اللغات البرمجية الأخرى حيث لا تستدعى إجرائية العرض باسمها مباشرةً وإنما تستخدم الإجرائية التالية لاستدعاء إجرائية العرض:

glutDisplayFunc(void*_func_ (void))

حيث يمرر له كوسيط اسم إجرائية العرض التي تم وضع كتلة الرسم فيها مع ملاحظة أن هذه الإجرائية يجب أن تكون بدون وسطاء. وتستدعى هذه التعليمة للمرة الأولى في الـ main ويمكن استدعاؤها عند الحاجة لإعادة الرسم .

سنلاحظ لاحقاً أن هذا الأسلوب سيستخدم كثيراً في استدعاء العديد من الإجرائيات .

إجرائية إعادة رسم شاشة الإظهار:

تستدعى هذه الإجرائية من أجل إعادة رسم شاشة الإظهار وذلك في عدة حالات هي :

١- بعد إنشاء نافذة الإظهار مباشرة وقبل استدعاء إجرائية العرض .

٢- عند تغيير حجم النافذة بتكبيرها أو تصغيرها .

٣- عند تغيير موقع النافذة بتحريكها من مكانها .

تستدعى في الإجرائية `main` بالشكل التالي :

```
glutReshapeFunc(void*func (int h, int w));
```

حيث تتتبع الإجرائية **glutReshapeFunc** الحالات السابقة ، وبمجرد حدوث أحدها تستدعي إجرائية إعادة الرسم، وعلى الرغم من وجود وسيطين لإجرائية إعادة الرسم إلا أنهما لا يمرران لها عند استدعائها بالشكل السابق. فلو اعتبرنا أن اسم إجرائية إعادة الرسم `Reshape` عندها تستدعى في الـ `main` بالشكل التالي :

```
glutReshapeFunc(Reshape);
```

أما جسم الإجرائية `Reshape` فيتضمن مجموعة تعليمات متعلقة بنظام الإحداثيات المستخدم، أنواع التحويلات المراد استخدامها في التطبيق والتي سنتعرف عليها لاحقاُ.

مثال٢:

سنعيد كتابة المثال السابق باستخدام الشكل النظامي والصحيح لاستدعاء إجرائية العرض وتهيئة النافذة .

Example 1-2 : White Rectangle on a Black Background

```
#include <GL/gl.h>
#include <GL/glut.h>

void display(void)
{
   /*  clear all pixels  */
   glClear (GL_COLOR_BUFFER_BIT);

/*  draw white polygon (rectangle) with corners at
 *  (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)
 */
    glColor3f (1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
        glVertex3f (0.25, 0.75, 0.0);
    glEnd();

/*  don't wait!
 *  start processing buffered OpenGL routines
 */
    glFlush ();
}
void init (void)
{
/*  select clearing (background) color       */
    glClearColor (0.0, 0.0, 0.0, 0.0);
}
void reshape(int w, int h)
{
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();         /* clear the matrix */
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}
/*
 *  Declare initial window size, position, and display mode
 *  (single buffer and RGBA).  Open window with "hello"
 *  in its title bar.  Call initialization routines.
 *  sets the reshape callback for the current window.
 *  Register callback function to display graphics.
 *  Enter main loop and process events.
 */
void main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("hello");
    init ();
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutMainLoop();
}
```

مثال عملي:

سنقوم الآن بتنفيذ خوارزمية DDA لرسم الخط المستقيم باستخدام الـــ OpenGL :

Example 1-3 : Algorithim DDA with OpenGL

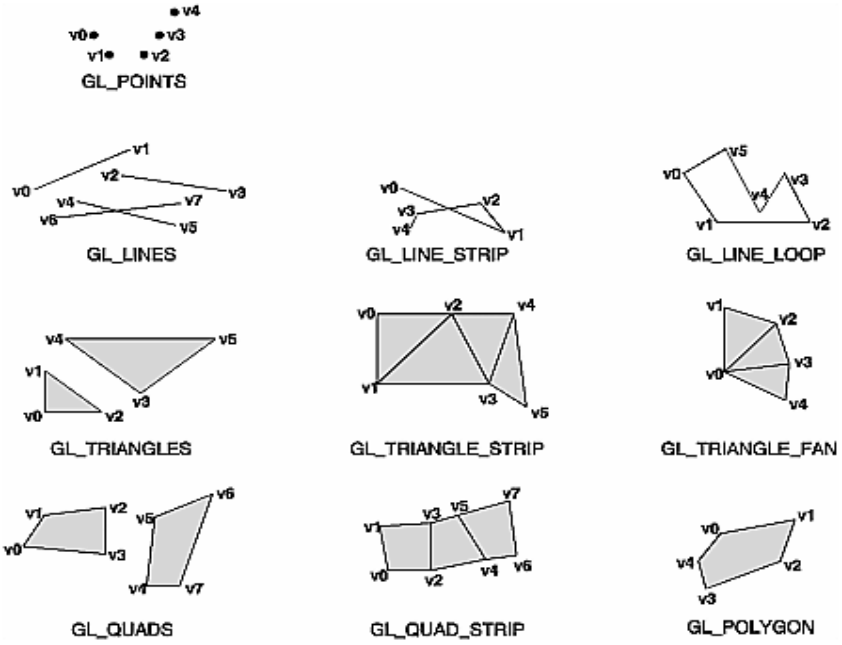
```
#include "stdafx.h"
#include <GL/gl.h>
#include <GL/glut.h>

void lineDDA ()
{
 glColor3f (1.0, 1.0, 1.0);
 int xa, int ya, int xb, int yb;
     xa = 0؛ ya = 0؛ xb = 50؛  yb = 50;
 int dx = xb - xa,   dy = yb – ya;
 int steps, k;
 float xIncr, yIncr, x = xa, y = ya;

 if (abs(dx)>abs(dy))
     steps = abs(dx);
 else
     steps = abs(dy);

 xIncr = dx / (float) steps;
 yIncr = dy / (float) steps;

 glBegin(GL_POINTS) ;
   glVertex2i (x, y);//as drawPixel (ROUND(x), ROUND(y))
   for (k=0; k<steps; k++)
   {
     x += xIncr;
     y += yIncr;
     glVertex2i (x, y);//as drawPixel (ROUND(x), ROUND(y))
   }
 glEnd();
 glFlush();
}
void init(void)
{
/*       select clearing (background) color  */
glClearColor (0.0, 0.0, 0.0, 0.0);
glClear (GL_COLOR_BUFFER_BIT);
/*  initialize viewing values   */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0.0, 100, 0.0, 100, -1.0, 1.0);
}
void main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (400, 400);
glutInitWindowPosition (100, 100);
glutCreateWindow ("lineDDA");
init();
glutDisplayFunc(lineDDA);
glutMainLoop();
}
```

لاحظ أن الثابت الممرر للتعليمة glBegin لم يكن GL_POLYGON وإنما ثابت آخر هو GL_POINTS وعندها سيتم رسم نقاط فقط ، ولكن هذه النقاط ستشكل في النهاية خط مستقيم بحسب خوارزمية DDA .

هناك العديد من الثوابت التي يمكن أن تمرر أيضاً ، والتي يظهر نتيجة لها أشكال هندسية مختلفة وهي على سبيل المثال :

الثابت	ماذا يعني
GL_POINTS	إظهار مجموعة نقاط منفصلة عن بعضها
GL_LINES	يتم الوصل بين كل نقطتين متتاليتين لإظهار خط مستقيم
GL_LINE_STRIP	إظهار مجموعة خطوط مستقيم متصلة مع بعضها
GL_LINE_LOOP	إظهار مجموعة خطوط مستقيمة متصلة مع بعضها ،مع وصل النقطة الأولى بالأخيرة وإظهار شكل مغلق .
GL_TRIANGLES	إظهار مثلث أو أكثر منفصلة عن بعضها ، بحيث كل ٣ نقاط تشكل مثلث .
GL_TRIANGLE_STRIP	إظهار مجموعة مثلثات ملتصقة مع بعضها
GL_TRIANGLE_FAN	إظهار مجموعة مثلثات تتصل مع بعضها برأس وضلع
GL_QUADS	إظهار شبه منحرف أو أكثر منفصلة عن بعضها
GL_QUAD_STRIP	مجموعة أشباه منحرفة ملتصقة مع بعضها
GL_POLYGON	إظهار مضلع ما

وسنستعرض الأشكال الناتجة عن كل ثابت :



الفصل الأول

تعريف بمكتبة الـــ OpenGL

تمرين١ :

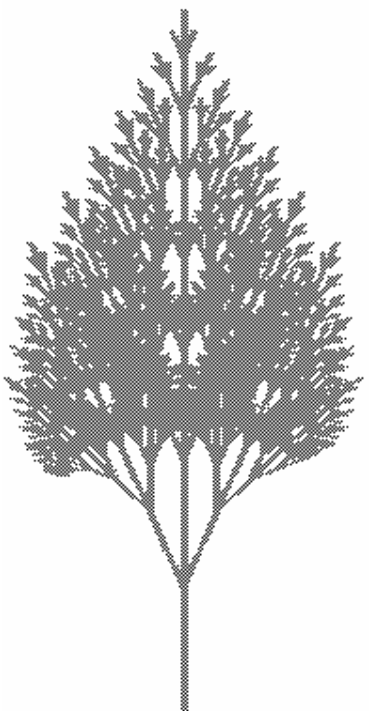
لنتذكر معاً خوارزمية الرسم التقطيعي fractal (الشجرة)التي تعرفنـــا عليهـــا فـــي كتـــاب البرمجة٢ ، حيث كنا قد اعتبرنا أن الشجرة عبارة عن مجموعة أغصان تتكرر بـــشكل مـــصغر (كفروع) أكثر من مرة لتعطي الشكل الكلي.

وقد كتبنا تطبيق بلغة الباسكال محققين رسم خوارزمية الشجرة :

```
Program tree_GU;
Uses GU;

Procedure tree(ng: integer; l,ti: real)
Var a,px,py: real;

Procedure branch(ox,oy:real; nr: integer; l,t: real);
Var Px1,Px2,Py1,Py2 : real;
Begin
      If( (nr <> 0) and (l>0))then
       Begine
            Px1 := ox; Py1:= oy;
            Px2 := ox + l*cos(t);
            Py2 := oy + l*sin(t);
            nr := nr –l;
            GLULine(Px1,Py1,Px2,Py2);
            branch(Px2,Py2,nr,l-l*0.45,t+ti);
            branch(Px2,Py2,nr,l-l*0.15,t);
            branch(Px2,Py2,nr,l-l*0.45,t-ti);
       end;
end; (*branch*)
begine
a: pi/2;
px := 0.5; py := 0;
branch(px,py,ng,1,a);
end;
Begine
(****************************************)
(***     Graphic Mode Initialisation     ***)
```

```
GUInitGraph;
GUWindowView(0,0,1,1,50,50,450,450);
tree(8,0.2,pi/8);
GUText(0.05,0.9,’Fractal Example.’);
readln;
GUCloseGraph;
End.
```

فحصلنا على الخرج التالي :

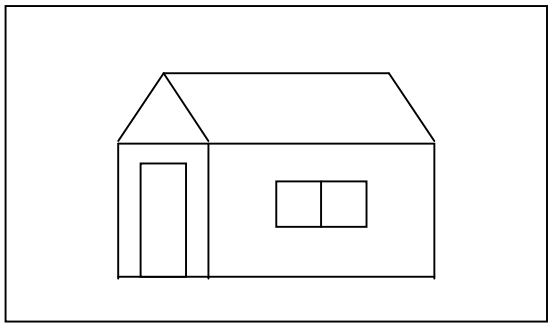


والمطلوب كتابة تطبيق نحقق فيه خوارزمية الشجرة مستخدمين تعليمات مكتبة الـــ OpenGL ، و لغـــة البرمجة C++ .

يمكنك الرجوع إلى كتاب البرمجة٢ لتذكر وحدات GU التي استخدمناها هنا .

تمرين ٢:

اكتب تطبيق باستخدام مكتبة الـــ OpenGL بحيث يظهر في هذا التطبيق كوخ له الشكل التالي:



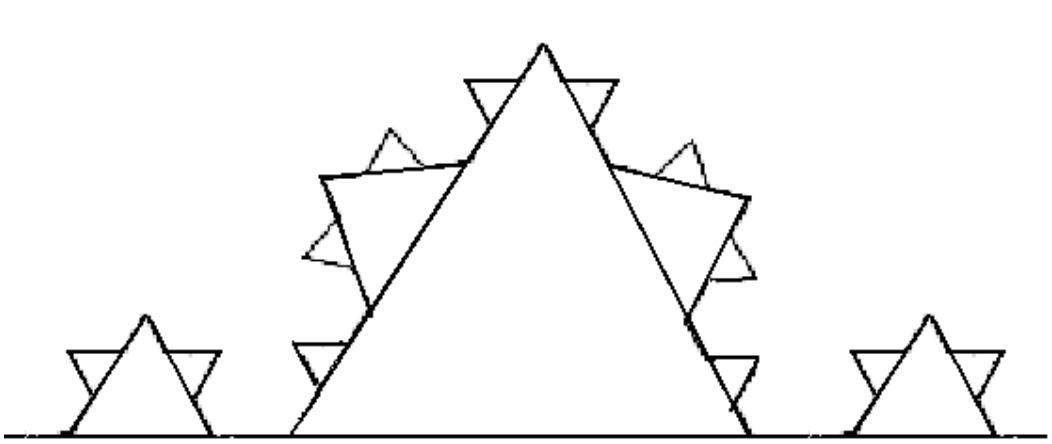
مستفيداً من التعليمات التي ذكرناها .

تمرين٣ :

اكتب برنامج لمعرفة الوضع النسبي لقطعتين مستقيمتين إذا كانتا متوازيتين أم متقاطعتين أم متخالفين، وفـــي حال تقاطعهما نريد معرفة إحداثيات نقطة التقاطع ووضع نقطة التقاطع (تنتمي إلى القطعتين ، تقع على واحدة من القطعتين وحامل القطعة الثانية، تقع على حامل القطعتين).

تمرين ٤ :

اكتب تطبيق يظهر الشكل التالي



مستخدماً أسلوب البرمجة العودي ، و تعليمات الـــ OpenGL للرسم .

الفصل الثاني
التفاعلية مع المستخدم

مقدمة:

تستطيع الـ OpenGL تقديم تطبيقات رسوميّة رائعة ،وتجسيد النماذج الواقعية بما توفره لمستخدميها من أشكال هندسية وكائنات رسومية جاهزة تلبي كل الاحتياجات التي يتطلبها أي نموذج رسومي . إلا أن التطبيقات الرسومية لا تكتسب أهميتها من قدرتها على رسم مشهد أو نموذج لمجرد تجسيد صورة فحسب ، بل تكتسبها عندما تكون قادرة على التفاعل مع المستخدم والاستجابة لأوامره . ويعتبر مجال الألعاب الحاسوبية مجالاً واسعاً جداً لاستخدام التطبيقات الرسومية القادرة على الاستجابة لأوامر المستخدمين، فإذا كانت التطبيقات الرسومية التي تقدمها الـ OpenGL غير قادرة على دخول هذا المجال فإنها ستبقى بعيدة عن الاستخدام ، لذلك كان لابد للـ OpenGL أن تمتلك القدرة على التفاعل مع المستخدمين والاستجابة لأوامرهم التي غالباً ما تصدر من الطرفيات المحيطية للحاسوب كالفأرة ولوحة المفاتيح ...الخ . وبالفعل استطاعت الـ OpenGL تحقيق أمر التفاعلية بالاستعانة بالمكتبة الخدمية GLUT التي قدمت تعليمات و إجرائيات تستجيب للأحداث الصادرة عن الفأرة ولوحة المفاتيح و ...الخ . إن الاستجابة التي يحققها التطبيق غالباً ما تكون نتيجتها الحركة على اختلاف أشكالها كالتدوير والانسحاب والتكبير والتصغير ، فمثلاً في التطبيقات الرسومية في مجال الألعاب (كسباق السيارات) هدف المستخدم من التفاعل مع التطبيق هو تحريك هذه السيارة نحو الأمام أو العودة فيها نحو الخلف و التفافها ...الخ . وتوابع الحركة هذه توفرها مكتبة الـ OpenGL .

ونلخص فيما مراحل التفاعلية والحركة:

١- يصدر أمر من المستخدم باستخدام الطرفيات المحيطية .
٢- يستجيب التطبيق لهذا الأمر (باستخدام التوابع التي توفرها المكتبة الخدمية GLUT) .
٣- النتيجة حركة ما في أشكال النموذج الذي يعرضه التطبيق (باستخدام توابع الحركة التي توفرها مكتبة الـ OpenGL) .

مثال١:

في هذا المثال سنركز على الاستجابة للأحداث التي تصدر عن الفأرة ولوحة المفاتيح بحيث :

- عند الضغط على الزر الأيمن للفأرة سيبدأ الشكل الظاهر بالدوران .
- عند الضغط على الزر الأيسر للفأرة سيتوقف الشكل الظاهر عن الدوران .
- عند الضغط على زر ESC في لوحة المفاتيح سيتم إنهاء التطبيق .

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

static GLfloat spin = 0.0;

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glPushMatrix();
   glRotatef(spin, 0.0, 0.0, 1.0);
   glColor3f(1.0, 1.0, 1.0);
   glRectf(-25.0, -25.0, 25.0, 25.0);
   glPopMatrix();
   glutSwapBuffers();
}
void spinDisplay(void)
{
   spin = spin + 2.0;
   if (spin > 360.0)
      spin = spin - 360.0;
   glutPostRedisplay();
}
void init(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glShadeModel (GL_FLAT);
}
void reshape(int w, int h)
{
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
void mouse(int button, int state, int x, int y)
{
   switch (button) {
      case GLUT_LEFT_BUTTON:
         if (state == GLUT_DOWN)
            glutIdleFunc(spinDisplay);
         break;
      case GLUT_MIDDLE_BUTTON:
      case GLUT_RIGHT_BUTTON:
         if (state == GLUT_DOWN)
            glutIdleFunc(NULL);
         break;
      default:
         break;
   }
}
```

```
void keyboard (unsigned char key, int x, int y)
{
    if (key = 27)
         exit(0);
}
/*
 *  Request double buffer display mode.
 *  Register mouse input callback functions
 */
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowSize (250, 250);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMouseFunc(mouse);
   glutKeyboardFunc(keyboard);

   glutMainLoop();
   return 0;   /* ANSI C requires main to return int. */
}
```

سنبدأ بشرح التعليمات المتعلقة بالاستجابة لأحداث الفأرة ولوحة المفاتيح :

إجرائية الاستجابة لحدث الفأرة:

void mouse(int button,int state,int x,int y)

بارمتراتها هي :

A. Button : يحدد أي زر من أزرار الفأرة تم ضغطه ويأخذ هذا الوسيط إحدى قيم الثوابت التالية :

Button = GLUT_LEFT_BUTTON

Button = GLUT_RIGHT_BUTTON

Button = GLUT_MIDDLE_BUTTON

B. State : يحدد حالة زر الفأرة هل هو مضغوط أم لا :

State = GLUT_UP

State = GLUT_DOWN

C. X,Y إحداثيي الفأرة على الشاشة .

في الإجرائية الرئيسية الـــmain يستدعى التابع الذي يلتقط الأحداث التي تصدر عن الفأرة .

```
glutMouseFunc(mouse);
```

إجرائية الاستجابة لأحداث لوحة المفاتيح:

```
void keyboard (unsigned char key, int x, int y)
```

بارمتراتها هي :

A. char : يحدد أي المحرف الذي تم الضغط عليه في لوحة المفاتيح ، ويمكن أن يكون رقم يرمز للـــ ascii code الخاص بهذا المحرف ،كما في المثال السابق حيث تعاملنا مع الـــــ ascii code لمحرف الـــ ESC والذي يساوي القيمة ٢٧ .

B. X,Y إحداثيي الفأرة على الشاشة عندما تم ضغط المحرف في لوحة المفاتيح .

في الإجرائية الرئيسية الـــmain يستدعى التابع الذي يلتقط الأحداث التي تصدر عن لوحة المفاتيح .

```
glutKeyboardFunc(keyboard);
```

ملاحظة١:

١. رغم أن للإجرائيتين mouse و keyboard بارمترات إلا إنه لا تمـــرر لهـــا عنـــد اســـتدعاء الإجرائية glutKeyboardFunc ، glutMouseFunc .
٢. إن اسم الإجرائية ليس اسم محجوز، بمعنى أنه يمكن تغيير اسم الإجرائية كما تريد ولكن الشيء الأساسي هو عدد بارمترات الإجرائية و نوعها .

أما فيما يخص الحركة توجد تعليمات ضرورية لتحقيقها:

١. توابع الحركة : وقد استخدمنا هنا أحد هذه التوابع وهو تابع الدوران `glRotatef(spin, 0.0, 0.0, 1.0);` في إجرائية العرض والذي سيشرح مع باقي تعليمات التحريك كالانسحاب وغيره في الفصل القادم ، ولكن ما نركز عليه هنا هو طريقة جعل هذا التابع يعطي مفهوم الحركة بجعل زاوية الدوران والتي تتمثل بالمتحول `spin` ذو قيمة متغييرة .

٢. التابع glutSwapBuffers() : تتم عمليات الرسم عادةً على السطح الخلفي back buffer، وعند استدعاء glutSwapBuffers() ينقل المشهد المرسوم إلى السطح الأمامي front buffer حيث يظهر على الشاشة . والهدف تسريع عمليات الإظهار على الشاشة، ومنع حدوث التضارب. ويتم استدعاء عمليات رسم المشهد في لحظة معينة بين التابعين glClear، و glutSwapBuffers كما نرى من التابع display.

٣. `glPushMatrix()` : تابع ينسخ المصفوفة الحالية ويضيف النسخة في أعلى المكدس فوق المصفوفة السابقة.

٤. `glPopMatrix()` : تابع يتخلص من المصفوفة الحالية.

سيتم شرح عمل هاتين التعليمتين بالتفصيل في الفصل القادم.

إظهار الحركة بشكل مستمر:

١. يجب إجراء تغيير مستمر في قيمة زاوية الدوران `spin` التي تمرر للتابع `glRotatef` .

٢. إعادة رسم النافذة الحالية عند كل تغيير لزاوية الدوران .

يقوم التابع **`spinDisplay`** بتغيير في قيمة المتحول `spin` ، ومن ثم إعادة رسم النافذة الحالية من جديد لأخذ التغيير في قيمة الزاوية `spin` . ولإعادة رسم النافذة نستخدم التابع `glutPostRedisplay()` .

`glutPostReDisplay()` : تعمل كاستدعاء عودي لإجرائية العرض، حيث لا يتم استدعاء إجرائية العرض – كما تعلمنا في لغات البرمجة – باسمها مباشرة .

إن الإجرائية `spinDisplay()` تجري تغيير في قيمة المتحول `spin` مرة واحدة ، ولتغيير قيمة الزاوية بشكل مستمر يجب استدعاء التابع `spinDisplay()` عودياً . ولتحقيق ذلك نستخدم التابع `glutIdlFunc()` .

`glutIdlFunc()` : يستخدم لإظهار حركة مستمرة دون تدخل من المستخدم ، يمرر لهذا التابع بارمتر يأخذ إحدى القيمتين التاليتين :

- اسم الإجرائية التي تجري تغيير في متحولات تابع الحركة ، والتي تعيد رسم النافذة الحالية وهنا الإجرائية التي تمرر لهذا التابع هي `spinDisplay()` .
- القيمة `NULL` والتي توقف عمل التابع `glutIdlFunc()` وبالتالي تتوقف الحركة.

ملاحظة٢:

لو استدعينا التابع spinDisplay() مباشرة في إجرائية الماوس دون استخدام التابع glutIdlFunc() عندها ستظهر الحركة مع كل نقرة ماوس ، أما باستخدام التابع glutIdlFunc() فستظهر حركة مستمرة بمجرد النقر مرة واحدة على الماوس ، ولإيقاف هذه الحركة نضغط الزر الأيسر للماوس فيستدعى التابع glutIdlFunc مع البارمتر NULL .

glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB) : سبق وذكرنا هذه التعليمة ولكن كان الثابت الذي نستخدمه GLUT_SINGLE حيث كان التطبيق يظهر أشكال رسومية ساكنة ، أما الآن فنستخدم الثابت GLUT_DOUBLE لأن الأشكال المرسومة متحركة والمقصود بالكلمة DOUBLE أننا نتعامل مع مخزنين مؤقتين للنقط المرسومة two Buffer. حيث يتم إظهار النقاط الموجودة في المخزن المؤقت الأول ، في الوقت الذي تكون النقاط في المخزن الثاني يتم رسمها وحالما يتم الانتهاء من رسم النقاط الموجودة في المخزن المؤقت الثاني يتم إظهارها والتبديل مع نقاط المخزن الأول الذي توضع فيه نقاط جديدة للرسم.

إنشاء القوائم:

يمكن للمستخدم أيضاً أن يتفاعل مع التطبيقات من خلال القوائم المنسدلة حيث يختار عنصر ما من القائمة ، ويختار المستخدم أمر ما (عنصر ما) من القائمة باستخدام الفأرة . كما يمكن التحكم بعدد العناصر في القائمة و باختلاف الخيارات تختلف النتائج .

تقدم الـ GLUT مجموعة تعليمات لإنشاء القوائم سنشرحها فيما يلي :

١. إجرائية إنشاء القوائم :

```
int glutCreateMenu(void (*func)(int value));
```

تعيد هذه التعليمة رقم يمثل رقم القائمة ، أما بارمتر هذه الإجرائية فهو إجرائية أيضاً بارمترها value وهو يأخذ قيمة العنصر الذي يختاره المستخدم من القائمة .

٢. إجرائية لإضافة العناصر للقائمة :

```
void glutAddMenuEntry(char *name, int value);
```

تقوم هذه الإجرائية بإضافة عنصر جديد إلى القائمة

بارمترات هذه الإجرائية :

a. Name : اسم العنصر الذي ستتم إضافته للقائمة، حيث سيختاره المستخدم من القائمة.

b. Value: رقم العنصر في القائمة ، والذي سنستفيد منه لمعرفة الحدث المراد تنفيذه .

٣. إجرائية لربط القائمة المنسدلة بأحد أزرار الفأرة :

```
void glutAttachMenu(int button);
```

حيث تتسدل القائمة بمجرد النقر على الزر الذي ارتبطت به القائمة . يأخذ هذا البارمتر إحدى القيم التالية :

```
Button = GLUT_LEFT_BUTTON
Button = GLUT_RIGHT_BUTTON
Button = GLUT_MIDDLE_BUTTON
```

ملاحظة ٣:

في حال ربط زر الفأرة مع القوائم المنسدلة ، و استخدامه لتنفيذ أحداث تخص إجرائية الفأرة فسيلغى تفعيل هذا الحدث المرتبط مع إجرائية الق... ويبقى أثره فقط في إجرائية .... ، لذلك يجب الانتباه على عدم استخدام الزر في الإجرائيتين بنفس الوقت .

مثال٢:

تستدعى جميع الإجرائيات السابقة ضمن جسم الإجراء الأساسي main .

```
void main(int argc, char** argv)
{
      . . .

      glutCreateMenu(menu)
      glutAttachMenu(GLUT_LEFT_BUTTON);
      glutAddMenuEntry("draw with red color",1);
      glutAddMenuEntry("draw with blue color",2);
      glutAddMenuEntry("draw with green color",3);

      . . .
}
```

الأيسر للفأرة لأنه الزر الذي ارتبطت معه القائمة المنسدلة . أما بالنسبة لإجرائية إنشاء القائمة فلاحظ البارمتر الممرر لها **menu** هو عبارة عن اسم إجرائية يتم فيها الاستجابة للحدث المرتبط مع العنصر الذي

يختاره المستخدم من القائمة .

سنتناول في هذا المثال إجرائيتي العرض والقائمة المنسدلة ، حيث نرسم في إجرائية العرض مضلع ، وباختيار أحد عناصر القائمة يتحدد لون خط الرسم فيظهر الشكل باللون الذي تم اختياره .

```
void display(void)
{
    glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
        glVertex3f (0.25, 0.75, 0.0);
    glEnd();
    glFlush ();
}

void menu(int value)
{
   if (value==1)  // the first item is selected, and
                  // the color is red
     {
       glColor3f(1.0,0.0,0.0);
       glClear(GL_COLOR_BUFFER_BIT);
       glutPostRedisplay();
     }
  else if (value==1)  // the second item is selected, and
                      // the color is blue
     {
       glColor3f(0.0,0.0,1.0);
       glClear(GL_COLOR_BUFFER_BIT);
       glutPostRedisplay();
     }
else if (value==2)  // the third item is selected, and
                    // the color is green
     {
       glColor3f(0.0,1.0,0.0);
       glClear(GL_COLOR_BUFFER_BIT);
       glutPostRedisplay();
     }
}
```

تعليمات أخرى خاصة بالقوائم المنسدلة :

١. إضافة قائمة جزئية :

void glutAddSubMenu(char *name, int menu);

بارمتراتها : اسم العنصر الذي ترتبط به القائمة الجزئية، رقم القائمة التي تتضمن القائمة الجزئية.

٢. تغيير اسم العنصر في القائمة :

void glutChangeToMenuEntry(int entry, char *name, int value);

بارمتراتها: رقم العنصر الذي سيتم تغييره في القائمة، الاسم الجديد للعنصر، القيمة الجديدة المرتبطة بهـذا العنصر .

غالباً ما نرى في القوائم المنسدلة تغيير اسم العنصر بمجرد اختياره ، وذلك عندما يكون لهذين العنـصرين أثريين متعاكسين ، أو إذا كان أثر العنصر الأول لا يظهر حتى يتم تنفيذ الحدث المرتبط بالعنـصر الثـاني والعكس بالعكس .

مثال٣:

إذا كنا نريد تكبير وتصغير شكل ما فإننا لا نضيف إلى القائمة عنصرين الأول للتكبير وآخر للتصغير وإنما نضيف عنصر واحد كالتالي:

```
glutAddMenuEntry("zoom out",1);
```

وبإجرائية الـ menu عند اختيار هذا العنصر نغيره ليصبح له الأثر المعاكس كالتالي:

```
void menu(int value)
{
if (value==1)
{
   glutChangeToMenuEntry(1,"zoom in",2);//change the item
   //--- add the appropriate response ---- //
 }
else if (value==2)
{
   glutChangeToMenuEntry(1,"zoom out",1);
   //--- add the appropriate response ---- //
}
}
```

٣. إزالة وحذف عنصر من قائمة:

void glutRemoveMenuItem(int entry);

٤. فك الربط مع زر الفأرة :

void glutDetachMenu(int button);

٥. هدم القائمة المنسدلة :

void glutDestroyMenu(int menu);

تمرين:

اكتب تطبيق تستخدم فيه القوائم المنسدلة ، بحيث يظهر في هذا التطبيق مضلع مرسوم بلون أبيض ، ثم يتغيَّر لونه بحسب الخيارات في القائمة ، وأضف خيارات جديدة للقائمة بحيث نختار من القائمة العنصر مثلث فيظهر لدينا شكل المثلث و غيرها من الأشكال التي تعرفنا عليها سابقاً .

الفصل الثالث

التحويلات والحركة

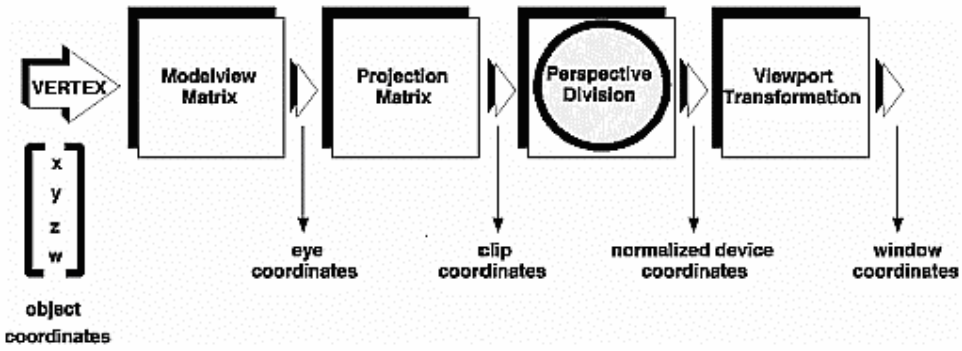
مقدمة:

إذا ما نظرنا في أية صورة بديعة المنظر ورائعة التصميم نظرة المحلل، لوجدناها تتألف من عـدة أشـكال هندسية بسيطة.

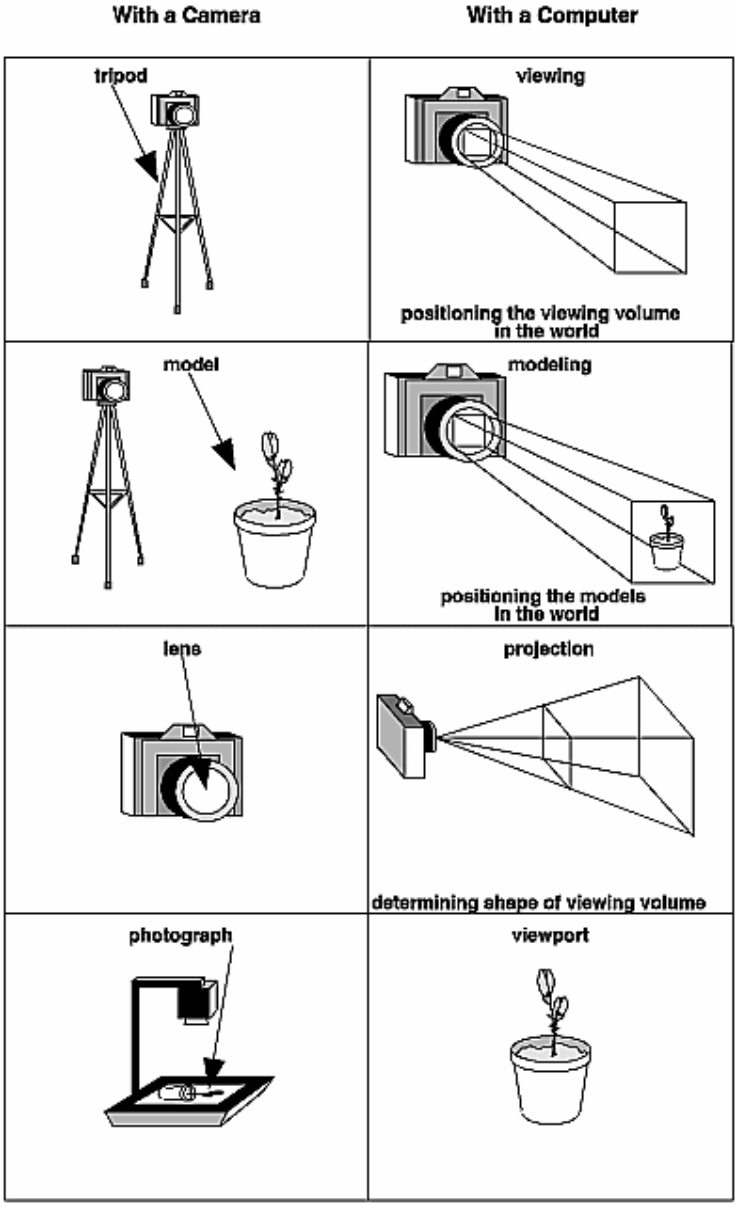
تستطيع الـ OpenGL تجسيد الصور الواقعيّة بما توفره من أشكال هندسية بسيطة تطبق عليها العديد مـن العمليات لتظهر بالشكل الملائم والمشابه للواقع . وهذه العمليات تشبه إلى حد كبير تلك التي نجريها عنـدما نريد التقاط صورة ما باستخدام كاميرا التصوير. ويمكن تقسيم هذه العمليات إلى ثلاث مراحل أساسية هي:

١. التحويلات وتتضمن:
   - تحويلات النمذجة: التي تتحكم بمكان ظهور الأشكال وحجمها.
   - تحويلات الرؤية: التي تتحكم بمكان توضع عين الناظر أو كاميرا التصوير و الجهة التي يتم النظر منها.
   - تحويلات الإسقاط: ومن خلاله يتحدد فيما إذا كـان النمـوذج المرسـوم ذو مـسقط منظوري أو مسقط متوازي.
٢. نافذة الرؤية: المسؤولة عن إظهار الأشكال بما يتناسب مع إحداثيات نافذة الرؤية.
٣. القص والقطع: التي تحدد الأشكال أو الأجزاء التي سيتم إظهارها ضمن النموذج البياني.

ويوضح لنا الشكل التالي المراحل المتلاحقة لتنفيذ هذه العمليات :



مقارنة بين العمليات في الحاسوب لرسم نموذج رسومي وتلك التي ننفذها في كاميرا التصوير:



لتحقيق تحويلات النمذجة، الرؤية، المنظور يجب إنشاء مصفوفة M ببعد 4X4 ومن ثم إجراء عملية ضرب لهذه المصفوفة بكل نقطة من نقاط الشكل v. M = v' (يجب أن لا ننسى أن العقدة الواحدة تتألف من أربع مركبات (x, y, z, w) حيث تأخذ w القيمة واحد عادةً عندما تكون الأشكال ذات موقع محدد، وقيمة تساوي الصفر عندما تقع الأشكال في اللانهاية، وبالنسبة لـــ Z فتأخذ في البعد الثنائي القيمة صفر .

طبعاً ليس من الضروري أن تخضع نقاط الشكل إلى كل هذه التحويلات وإنما نطبق التحـــويلات اللازمـــة لإظهار الصورة بالشكل المناسب .

ولنبدأ الآن بشرح كل نوع من هذه التحويلات ... .

▫ تحويلات النمذجة Modeling Transformations:

تستخدم هذه التحويلات للتحكم بمكان ظهور الأشكال وطريقة توضعها (مائلة، مـــستقيمة ، مقلوبة) بالإضافة إلى تحديد حجم الأشكال. ويمكن تطبيق هذه التحويلات على النماذج التي تحتوي أشكال ثنائية البعد والنماذج التي تحتوي أشكال ثلاثية البعد .

التحويلات والعلاقات الجبرية الممثلة لها في جملة الإحداثيات ثنائية البعد :

١. التقييس أو التحجيم : Scale

وفيه نستطيع تكبير أو تصغير أي شكل مهما كان معقداً إذا استطعنا تكبير أو تصغير إحداثياته وذلـــك بنفس النسبة.

تحيل بأنك تقف أمام برج عالي وتنظر اليه، تستطيع من خلال توابع التحويل أن تقوم بتقصير البـــرج ليتلاءم مع الارتفاع والعرض المرغوبين.

أما المعادلات الناظمة لهذه العملية فهي بسيطة، عبارة عن ضرب احداثيات أي نقطة بنفس النسبة:

- $x0 = x * sx$
- $y0 = y * sy$

٢. الدوران : Rotation

في هذا التحويل نستطيع أن نقوم بتدوير أي شكل كما نشاء، وفي أي جهة نشاء، كما نستطيع الـــتحكم بزاوية الدوران، فتخيل معي لو أننا ننظر إلى جسم سيارة ما، مثبتة على سطح قابل للدوران، نستطيع باستخدام توابع الدوران أن نقوم بتحريك هذا السطح الدوراني بالزاوية المرغوبة، بالتالي نستطيع رؤية هذه السيارة من كافة الجوانب وكأننا بتطبيق هذه المعادلات نقوم بعرض أنيق لسيارة فخمة كي يتمكن الرائي من أي مكان يقف فيه من رؤية السيارة بشكل كامل.

أما التوابع الناظمة لعملية الدوران فهي:

- $x0 = x * \cos\theta + y * \sin\theta$
- $y0 = y * \sin\theta + y * \cos\theta$

حيث θ تمثل زاوية الدوران.

٣. الانسحاب : Translation

تخيل لو أن لديك كرة في صندوق وأردت أن تحاكي حركتها، بالتالي ما عليك سوى تحريكها علـــى المحاور الثلاثة، فإذا غيرت قيم المحور الأفقي X فقط على سبيل المثال، فتستطيع محاكاة تحريـــك الكرة بشكل أفقي، أما إذا غيرت قيم الانسحاب على المحاور الثلاثة بنسب عشوائية، بالتالي تـــرى

الكرة تنتقل من مكان لآخر ضمن هذا الصندوق، طبعاً مع الأخذ بعين الاعتبار أن قيم الانسحاب هذه يجب ألا تتجاوز حدود الصندوق.

أما توابع الانسحاب فهي:

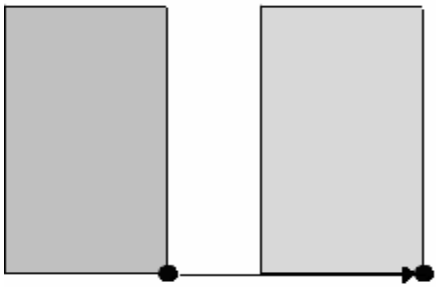
- `x0 = x + tx`
- `y0 = y + ty`

وبالنسبة للعلاقات الجبرية في جملة الإحداثيات ثلاثية البعد فإننا نأخذ بعين الاعتبار قيم الإحداثي z ، حيث تحسب قيمه كالإحداثيين x،y .

أما التوابع التي تقدمها الـ OpenGL لتحقيق هذه التحويلات فهي ثلاث توابع أساسية، وهي نفسها سواء في جملة الإحداثيات ثنائية البعد أو ثلاثية البعد :

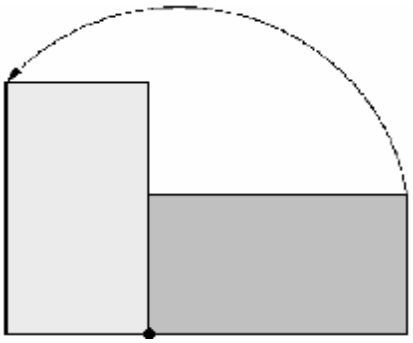
١. تابع الانسحاب glTranslatef(x,y,z) :

وهو التابع الذي يقوم بعملية الانسحاب ، حيث تشير البارمترات أن عمليات الانسحاب تتم على أحد المحاور x،y،z أو على جميع المحاور في نفس الوقت مع العلم أن تأثير الانسحاب على المحور z في البعد الثنائي لن يظهر ولذلك تأخذ z هنا القيمة صفر بشكل افتراضي ، ويمكن أن نجري انسحاب نحو اليمين أو اليسار ولفوق أو تحت وللأمام أو الخلف وذلك بالتحكم بإشارة القيمة المرتبطة بالبارمترات الممرر ة لهذا التابع بالسالب والموجب وعند جعل جميع القيم المتعلقة بالبارمترات x،y،z مساوية للصفر عندها لن يظهر أي تأثير لهذا التابع .



٢. تابع الدوران glRotatef(θ,x,y,z)

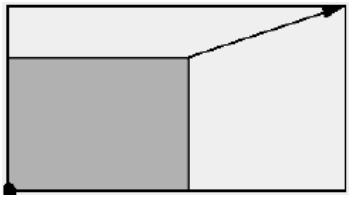
وهو التابع الذي يقوم بعملية الدوران ، حيث يشير البارامتر الأول θإلى زاوية الدوران ،أما البارمترات الثلاثة الأخرى فهي تسبب دوران حول أحد المحاور و نعطي المحور الذي نريد أن يدور حوله الشكل قيمة =١ أما باقي المحاور والتي لا نريد أن يدور الشكل حولها نعطيها قيمة = ٠ ، مع العلم أنه يمكن أن نعطي القيمة ١ لأكثر من محور وهذا يعني أن الدوران يتم على أكثر من محور وكذلك يمكن إجراء دورانات بجميع الاتجاهات وذلك بالتحكم بقيم الزاوية وحتى بإشارة الزاوية سالبة أو موجبة ولكن عند إعطاء الزاوية القيمة صفر عندها لن يحدث أي تأثير لهذا التابع .



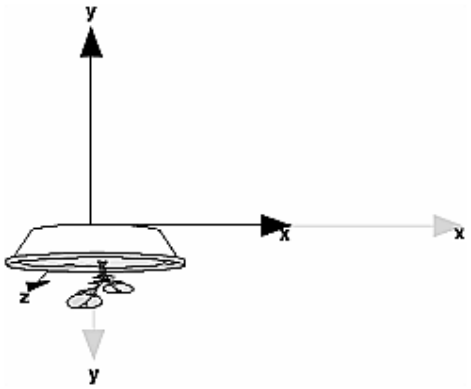
الشكل قبل التدوير الشكل بعد التدوير

٣. تابع التقييس glScalef (x,y,z):

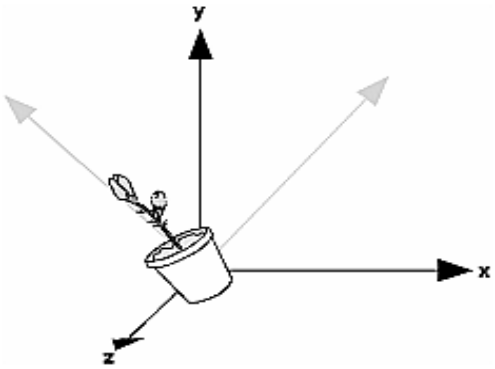
وهو تابع يقوم بتكبير أو تصغير الشكل على أحد المحاور أو على أكثر من محور كما أن لهـذا التـابع تأثير آخر حيث يمكنه أن يقلب الشكل وذلك بحسب قيم الـ z،y،x التي تمرر للتابع فإذا أعطي لأحـد هذه البارمترات قيمة أكبر من ١ فإن الشكل سيكبر ويزداد حجمه وإذا أعطي قيمه أقل من واحد فينقص حجم الشكل وعند إعطاء قيم سالبة تسبب قلب للشكل ولكن كما في حالة الانـسحاب فـإن التكبيـر أو التصغير في الأشكال ثنائية البعد لا يظهر تأثيره على المحور z .



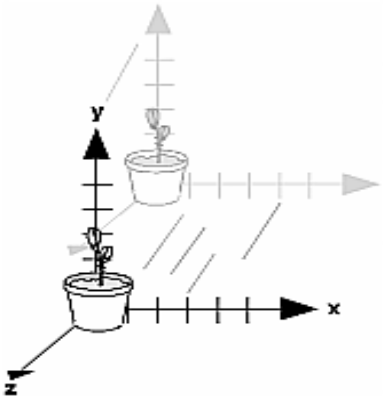
وبالنسبة لتأثيرات هذه التوابع على الأشكال ثلاثية البعد ، نأخذ الأمثلة التالية :



يمثل العكس على المحور y، والتكبير على المحور x

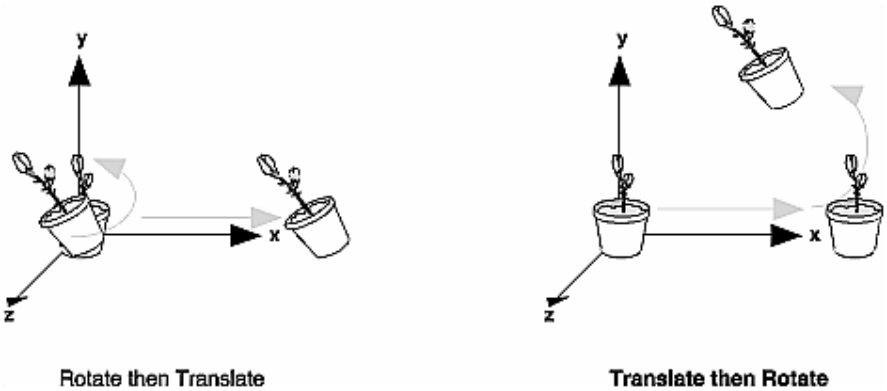


يمثل الدوران حول المحور z



يمثل الانسحاب على المحور z

ويمكن تركيب هذه التوابع مع بعضها بحيث نجري عملية انسحاب ثم تكبير ونلحقه بعملية دوران ولكن يجب أن ننتبه لموضوع هام جداً هو أننا إذا أجرينا تحويل١ ثم أجرينا تحويل٢ من نوع آخر فإن عكس هذين التحويلين بحيث نجري التحويل٢ ثم التحويل١ لن يعطي نفس التأثير على الشكل ولنأخذ كمثال الشكل التالي :



لاحظ أن إجراء الانسحاب بالأول ثم الدوران لا يعطي نفس تأثير تطبيق الدوران أولاً ثم الانسحاب .

مثال١:

تطبيق تحويلات النمذجة على الأشكال ثنائية البعد :

دوران الشكل حول نقطة ثابتة

لتحقيق دوران الشكل حول نقطة ثابتة يجب تطبيق التحويل التالي :

- `Start with identity matrix: C  I`
- `Move fixed point to origin: C  CT`
- `Rotate: C  CR`
- `Move fixed point back: C  CT-1`
- **`Result: C = TRT-1`**

سنأخذ كمثال على الأشكال ثنائية البعد المضلع والذي سنطبق عليه التحويلات اللازمة لتدوير الشكل حـــول نقطة ثابتة وطريقة تحقيقها في الـــ OpenGL .

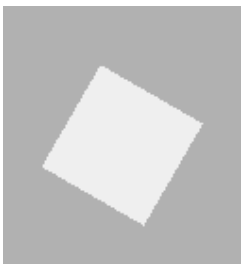
يأخذ المضلع الشكل التالي قبل إجراء أي تحويل

باعتبار أن أبعاده كالتالي (x1=0،y1=0،x2=25،y2=25)

وسنركز في هذا المثال على التعليمات في إجرائيتي العرض (display) وإجرائية التحديث وإعادة الرسم (reshape) .

```
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f (1.0, 1.0, 1.0);

glTranslatef(1.0,2.0,3.0); /* modeling transformation   */
glRotatef(30,0,0,1); /* modeling transformation         */
glTranslatef(-1.0,-2.0,-3.0); /* modeling transformation*/

 glBegin(GL_POLYGON);
   glVertex3f (0,0,0.0);
   glVertex3f (25,0, 0.0);
   glVertex3f (25, 25, 0.0);
    glVertex3f (0,25, 0.0);
 glEnd();
 glFlush ();
}

void reshape(int w, int h)
{
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();         /* clear the matrix */

   glOrtho(-50.0, 50.0, -50.0, 50.0, -50.0, 50.0);
}
```

وسيكون الخرج كما في الشكل التالي :



تحقق الــ OpenGL التمثيل المصفوفي بمجرد استدعاء التعليمات المسؤولة عن تحويلات النمذجة مع العلم أن الــ OpenGL تقدم نوعين من المصفوفات الأولى لتحقيق تحويلات النمذجة والثانية لتحقيق تحــويلات المنظور _ *التي سنتحدث عنها لاحقاً* _ وهما :

- **Model-View (GL_MODELVIEW)**
- **Projection (GL_PROJECTION)**

ولتحديد نوع التحويل المراد استخدامه تستدعى تعليمة واحدة للشكلين السابقين ولكن الذي يختلف هو الثابت الممرر لهذه التعليمة التي تأخذ الشكل التالي :

- **glMatrixMode(GL_MODELVIEW);**
- **glMatrixMode(GL_PROJECTION);**

وفي مثالنا السابق تــم اســتدعاء هــذه التعليمــة فــي إجرائيــة إعــادة الرســم `reshape` بالــشكل `glMatrixMode(GL_MODELVIEW);` وبما أن الثابت هو `GL_MODELVIEW` فهذا يعني أن التحويل الذي سيتم تطبيقه هو تحويل نمذجة بالإضافة إلى أن هذا الثابت يستخدم في تحويل الرؤية والــذي يستخدم دائماً مع تحويل النمذجة لذلك تم دمج هذين التحويلين بثابت واحد ، ولكن قبل إجراء أية عملية تحويل يجب تهيئة مصفوفة التحويلات **المصفوفة الواحدية** بالقيم الافتراضية أي بقيم تساوي الواحد وهذا ما تحققــه التعليمة **`glLoadIdentity`** حيث تستدعى من أجل مسح القيم المسندة لمصفوفة التحويل من جــراء تحويلات سابقة قد تكون أجريت في وقت من الأوقات وتحميل القيم الافتراضية للمصفوفة والتي تكون بشكل افتراضي رباعية (٤X٤) .

أما بالنسبة لتوابع التحويل في إجرائية العرض فإن الترتيب الذي تم استدعاؤها به يحقق التحويل المراد وهو دوران الشكل حول نقطة ثابتة .

```
glTranslatef(1.0,2.0,3.0);/*modeling transformation */
glRotatef(30,0,0,1);/* modeling transformation */
glTranslatef(-1.0,-2.0,-3.0);/* modeling transformation*/
Result: C = TRT-1
```

ملاحظة١:

تسمح الــ OpenGL بإجراء أي نوع من التحويلات وذلك بالسماح بتحميل أي قيم لمــصفوفة التحــويلات والتي سبق وذكرنا أن قيمها الافتراضية تساوي الواحد بالإضافة إلى إمكانية الضرب بمــصفوفات تحويــل مختلفة عوضاً عن استخدام توابع التحويلات كالانسحاب والدوران والتقييس التي تقــدمها الـــ OpenGL بحيث نحدد قيم هذه المصفوفات بحسب ما نرغب ويتم تحقيق هذا باستخدام التعليمتين التاليتين:

١. **`void glLoadMatrix{fd}(const TYPE *m);`** : وهي تقوم بنفس عمل التعليمة `glLoadIdentity` إلا أن الاخــتلاف بينهمــا أنــه فــي التعليمــة `glLoadMatrix` يجب تحديد مصفوفة التحويلات التي نريدها ثم نمررها كبارمتر لهــذه التعليمــة فــي حــين التعليمــة `glLoadIdentity` تحدد فيها مصفوفة التحويلات بشكل افتراضي وببعد 4X4 .

٢. **`void glMultMatrix{fd}(const TYPE *m);`** : تستخدم هذه التعليمة لإجـراء التحويل الذي نرغب بحيث نعرف مصفوفة تحويل ونعطيها قيم معينة ثم ننمررها كبارمتر لهذه التعليمة التي تقوم بدورها بإجراء عملية ضرب لهذه المصفوفة مع مصفوفة التحويل التي تم تعريفها سـابقاً ، وبالتالي فإن هذه التعليمة تنوب عن التعليمات التي تقـوم بعمليـات الانـسحاب والـدوران والتقيـيس `glTranslatef`، `glRotatef`، `glScalef` .

فعلى سبيل المثال يمكن تحقيق دوران الشكل حول نقطة ثابتة والذي رأيناه في المثال السابق بالشكل التالي :
أولاً نعرف مصفوفة التحويل الافتراضية بالبعد الذي نريده والقيم التي نريدها

$$M = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

```
glMatrixMode(GL_MODELVIEW);
glLoadMatrix (M);
glMultMatrixf(T);                  /* translation */
glMultMatrixf(R);                  /* rotation */
glMultMatrixf(T-1);               /* translation */
draw_the_object();
```

وعندما نريد استخدام مصفوفة التحويل الواحدية بدلاً من تعريف مصفوفة تحويل خاصة فإننا نستدعي التعليمة glLoadIdentity ثم نعرف مصفوفات التحويل الأخرى ونجري عمليات الضرب اللازمة ، كما في الشكل التالي :

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(T);                  /* translation */
glMultMatrixf(R);                  /* rotation */
glMultMatrixf(T-1);               /* translation */
draw_the_object();
```

مع ملاحظة أن R,T هما مصفوفتين يجب تعريفهما لإجراء تحويلي الانسحاب والدوران لجميع نقاط الشكل .

كل ما قمنا به حتى الآن هو تطبيق تحويلات النمذجة على شكل هندسي واحد، ولكن عندما نريد رسم نموذج ما يتألف من عدة أشكال هندسية متراكبة ومتجاورة عندها يجب تطبيق تحويلات النمذجة على هذه الأشـكال لتركيبها مع بعضها البعض بشكل صحيح ومتناسق.

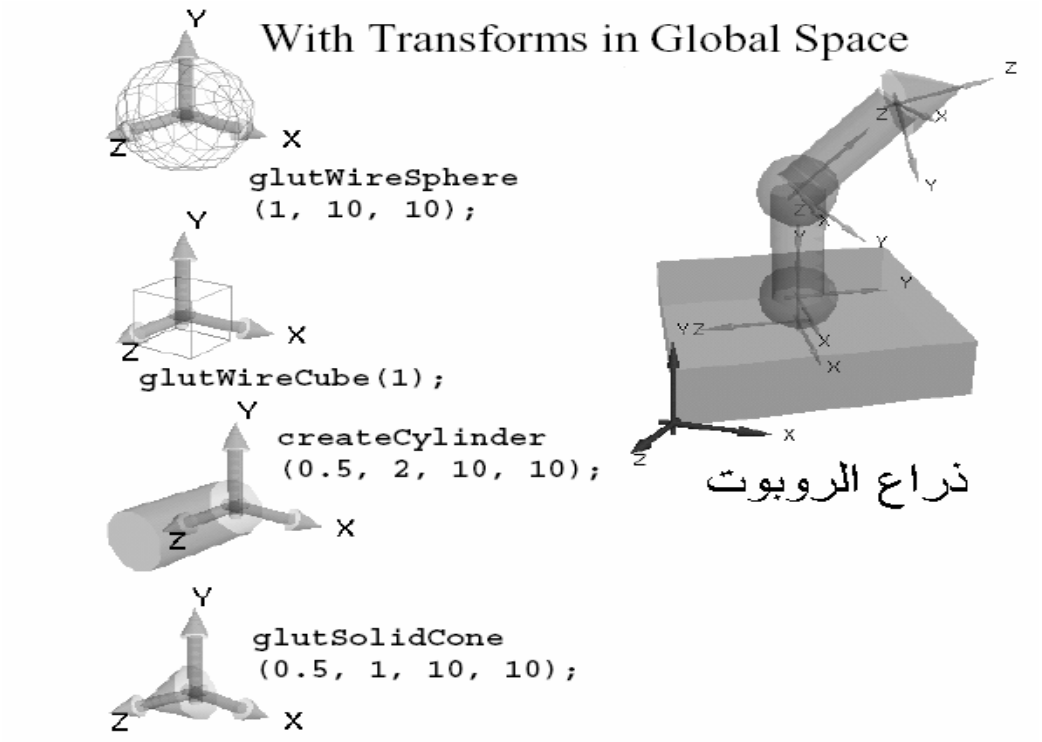
إن المبدأ العام لتركيب الأشكال مع بعضها يقوم على ربط الجملة الإحداثية لهذه الأشكالobject corrdinate بالجملة الإحداثية العالمية worldCorrdinate، بحيث نستطيع عندها تحديد البعد بينها، تراكبها، حجومهـــا بالنسبة لبعضها، وكل ذلك بإسقاط إحداثيات الأشكال على جملة الإحداثيات العالمية. وسنعرض مثال يوضح طريقة الاستفادة من توابع تحويلات النمذجة لتركيب الأشكال مع بعضها والحصول على كائن رسومي ما.

الفصل الثالث

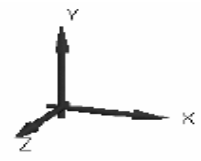
التحويلات والحركة

مثال٢:

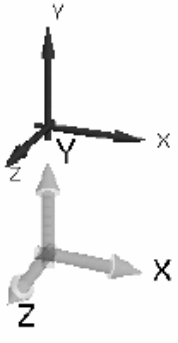
بفرض أن الكائن الرسومي عبارة عن ذراع لروبوت، وبفرض أن الأشكال المؤلفة لهذه الذراع هي: المكعب، جسم كروي، جسم أسطواني، جسم مخروطي:



- سنشرح بالتفصيل طريقة تركيب هذه الأشكال مع بعضها لتشكيل ذراع الروبوت :

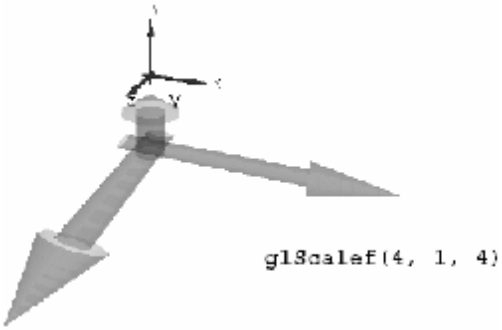


سنبدأ الرسم من مبدأ الإحداثيات للجملة التي تظهر باللون الأسود الغـــامق وهي جملة الإحداثيات العالمية بحيث نفترض أن جميع الأشكال مرســـومة استناداً لهذه الجملة.

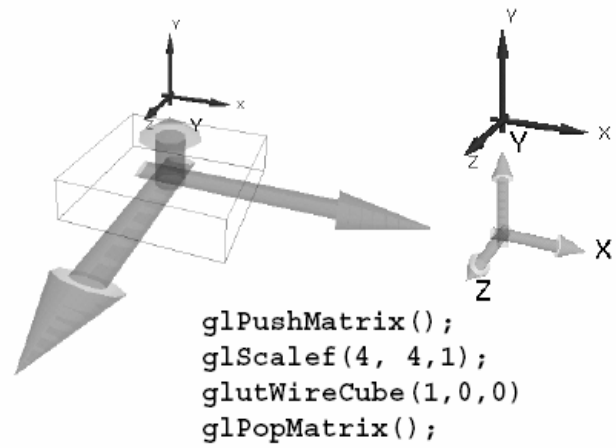


- نجري انسحاب على المحور y نحو الأسفل :

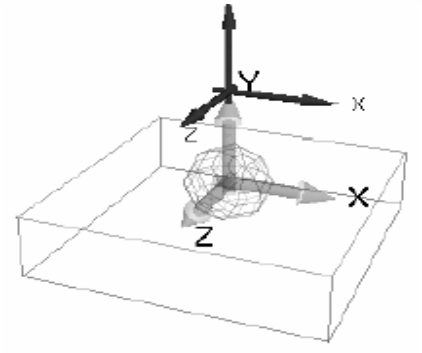
.glTranslate(0,-2,0)



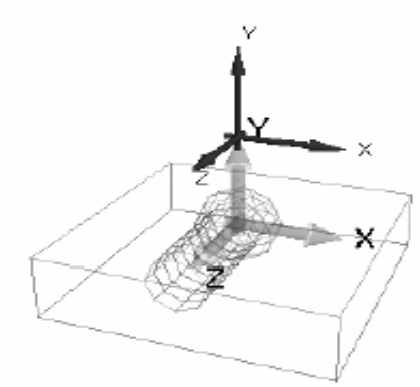
- نجري تكبير بنسبة (4,1,4).



- ثم نرسم المكعب ولكن بما أن عملية التكبير لا تناسب باقي الأشكال وإنما فقط من أجل القاعدة لذلك فإننا نستخدم التعليمتين glPushMatrix()، glPopMatrix() بحيث يتم استدعاء التعليمة glPushMatrix في البداية ، ثم تستدعى بعدها عملية التكبير ثم نرسم الشكل وبعدها نستدعي التعليمة glPopMatrix() للعودة للإحداثيات النظامية .

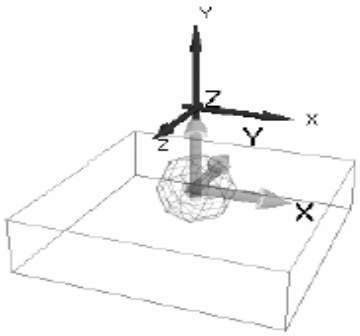


- انسحاب نحو الأعلى بمقدار 0.5 (نصف ارتفاع المكعب) ، ثم رسم الجسم الكروي بنصف قطر يساوي 0.6 .

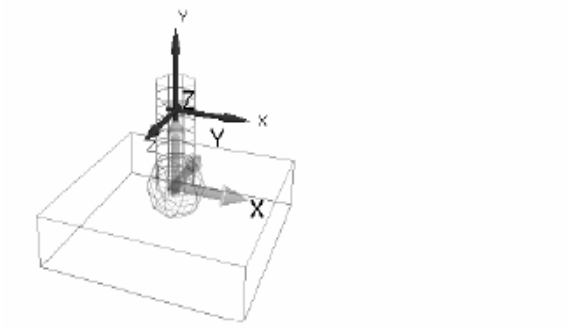


- والآن سنقوم برسم الجزء الأول من الذراع والذي نمثله بالاسطوانة، وإذا قمنا برسم هذه الإسطوانة مباشرة دون إجراء أي تحويل فستظهر بالشكل التالي :

ولكننا نريد أن تتجه هذه الذراع نحو الأعلى ، وهذا يتطلب إجراء عملية دوران حول المحور x بزاوية ولتكن تساوي 90- درجة ، فتأخذ المحاور الإحداثية الشكل التالي :




```
glRotatef(theta1, 1, 0, 0);
```

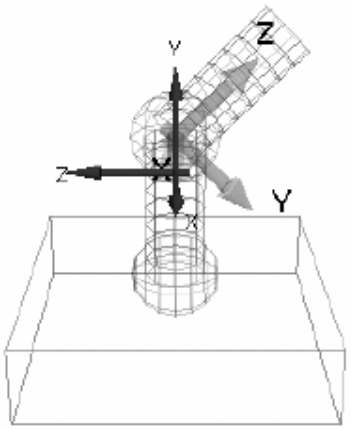


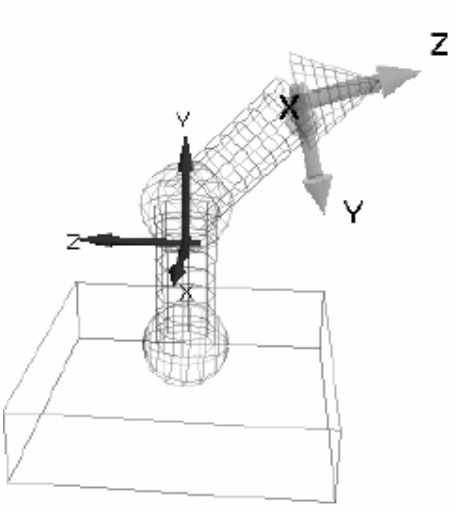

```
createCylinder(0.5, 2, 8, 8);
```

وبرسم الأسطوانة بحسب الإحداثيات الجديدة تظهر

لدينا الذراع بالشكل التالي:

- ثم نجري انسحاب على المحور y نحو الأعلى من جديد ونرسم الجسم الكروي الذي يمثل المفصل للذراع و لرسم الجزء الآخر من الذراع بشكل مائل نجري دوران بزاوية ما ولتكن تساوي ٤٥ درجة على المحور x ثم نستدعي تعليمة رسم الأسطوانة فيظهر لدينا الشكل النهائي للذراع .
- وأخيراً سنرسم الشكل المخروطي والذي يمثل المفصل الذي يربط بين الذراع والكتف بحيث نجري انسحاب للمحاور إلى أعلى الذراع، وندور بزاوية قدرها –٣٠ حول المحور x فيكون لدينا الشكل النهائي على النحو التالي :





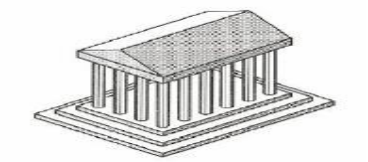
▫ تحويل الإسقاط Projection Transformations :

الإسقاط : هو تحويل مجموعة من النقاط من نظام إحداثيات بالبعد n إلى نظام إحداثيات ببعد أقل من n، وهنا سوف نحصر أنفسنا بالإسقاط من 3d إلى 2d ، حيث يستخدم الإسقاط للانتقال من جملة إحداثيات الرؤية إلى جملة إحداثيات جهاز العرض وهي 2d.

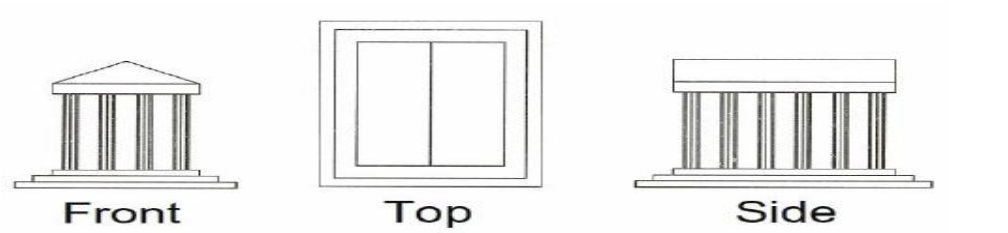
ينقسم الإسقاط إلى نوعيين أساسيين هما : **المتلاشي** perspective، **المطابق** parallel.

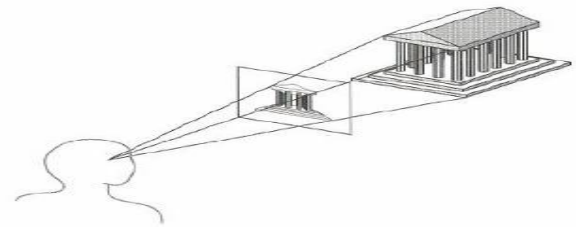
١. **الإسقاط المتلاشي أو المنظوري perspective** : حيث أن مجموعة المستقيمات المتوازية من الممكن أن تظهر غير متوازية على مستوي الإسقاط وتميل للالتقاء في نقطة واحدة تسمى نقطة التلاشي.
٢. **الإسقاط المطابق parallel:** حيث يكون مركز الإسقاط في اللانهاية، ويكون شعاع الإسقاط موازياً لسطح الإسقاط .

بفرض لدينا النموذج التالي :



فعند الإسقاط في المستوي المتوازي Orthographic يظهر لدينا النموذج بالشكل التالي:

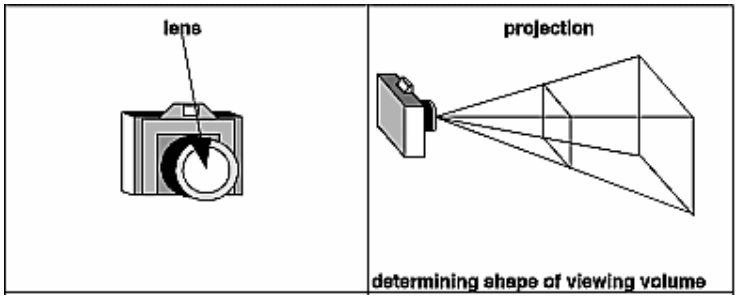




وباستخدام الإسقاط في الفراغ فإن الأجزاء الداخلية من النموذج ستظهر نوعاً ما، مما يجعله يبدو بشكل مجسّم.

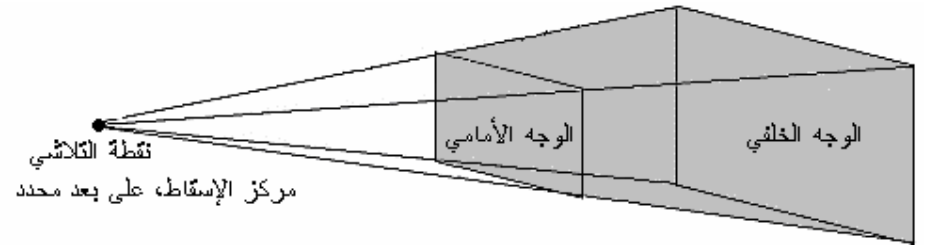
ويفيد الإسقاط في تحديد حيز الرؤية الذي نستطيع أن نرى كل ما يقع ضمنه، ولتوضيح هذا الأمر بشكل أكبر لنعود إلى تحويل الرؤية حيث استطعنا باستخدام هذا التحويل التحكم بمكان عين الناظر وتحديد الجهة التي ينظر إليها وذلك لمحاكاة ما يقوم به المصوِّر عندما يتخذ مكاناً مناسباً لالتقاط الصورة بكاميرا التصوير، ولكن

هذا ليس كل شيء، فالمصور بعد أن يحدد المكان الذي يقف به والجهة التي سيصور منها، يبدأ بالتحكم بعدسة كاميرا التصوير فيزيد من فتحة العدسة أو ينقصها وذلك لتحديد الحيز الذي يمكن للكاميرا تصويره. هذا الحيز سميناه بحيز الرؤية وهو ما يحدده لنا تحويل الإسقاط .



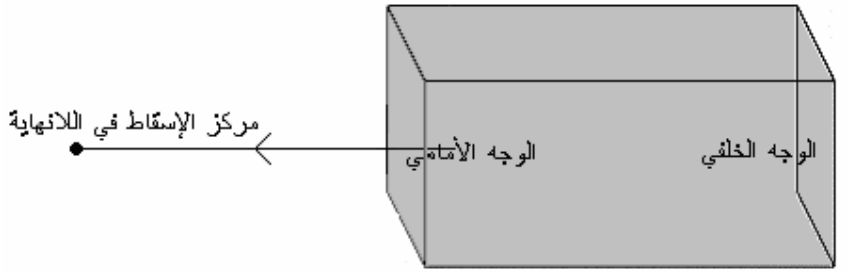
وبما أنه يوجد نوعين من الإسقاط فهذا يعني أنه يمكن تحديد نوعين من حيز الرؤية.

١. في حالة الإسقاط المنظوري :



حيث يكون حيز الرؤية عبارة عن جزع الهرم المحدد بالمستويين الأمامي والخلفي.

٢. في حالة الإسقاط المتوازي:



حيث يكون حيز الرؤية عبارة عن متوازي المستطيلات المحدد بالوجهين الأمامي والخلفي.

ويظهر الاختلاف بين نوعي الإسقاط السابقيين عندما نقوم بتقريب أو تبعيد كاميرا التصوير أي عندما نقوم بإجراء عملية zoom لعدسة الكاميرا ، ففي الإسقاط المنظوري أو المتلاشي يسبب إجراء عملية التقريب تكبير النموذج المصور، في حين تسبب عملية التبعيد تصغير النموذج. سنوضح ذلك بالشكل التالي:

▫ تحويل الرؤية Viewing Transformations:

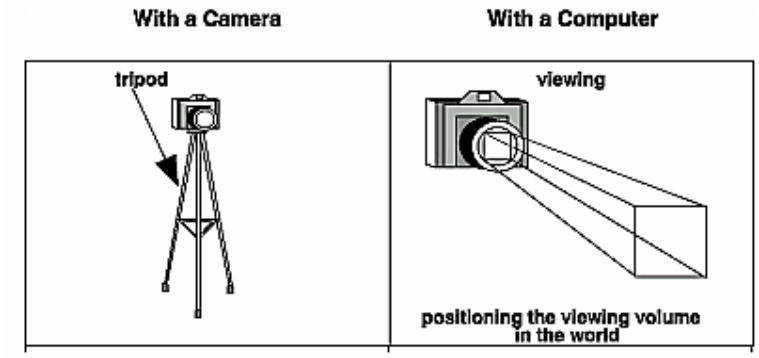
مقدمة:

ذكرنا سابقاً أن ما نريد تحقيقه في الـ OpenGL هو محاكاة عمل كاميرا التصوير وجعل المشهد يبدو وكأنه قد تم تصويره بالكاميرا ، واستطعنا باستخدام تحويلات النمذجة التحكم بطريقة إظهار الأشكال ضمن المشهد بحيث حددنا موقع كل شكل وحجمه بالإضافة إلى تدوير الأشكال التي أردنا أن تظهر بشكل مائل قليلاً. والآن نريد التحكم بطريقة عرض وإظهار الصورة ، حيث نريد إظهار المشهد بعدة أشكال مائل نحو اليمين، مائل نحو اليسار، مقلوب ...الخ. أو مثلاً لنعود إلى مثال السيارة المثبتة على سطح قابل للدوران، حيث استطعنا باستخدام توابع الدوران أن نقوم بتحريك هذا السطح الدوراني بالزاوية المرغوبة، وبالتالي رؤية هذه السيارة من كافة الجوانب، ولكن سنفترض أن هذا السطح ثابت وغير قابل للدوران، وأن الناظر هو من يقوم بالدوران حول السيارة والنظر إليها من كل الجوانب من الأسفل ومن الخلف ومن فوق والنظر إليها عن قرب أو عن بعد .

والسؤال المطروح هنا:

هل هنالك إمكانية تثبيت المشهد ولكن تطبيق تحريك لعين الناظر؟؟

عادةً نتخذ موقع لكاميرا التصوير من أجل التقاط صور جانبية أو تصوير المشهد من فوق أو من الأسفل لإظهاره بشكل فني جميل، ولتحقيق هذا الأمر في الـ OpenGL نستخدم نوع من التحويلات التي تدعى **تحويلات الرؤية** والتي تسمح لنا بالتحكم بطريقة عرض المشهد ككل بحيث نحدد الموقع الذي يتم فيه النظر إلى المشهد من الأعلى أو الأسفل أو من الجانب مع العلم أن تأثير تحويلات الرؤية لا تظهر إلا في النماذج ثلاثية البعد على عكس تحويلات النمذجة التي تؤثر في النماذج ثنائية وثلاثية البعد .



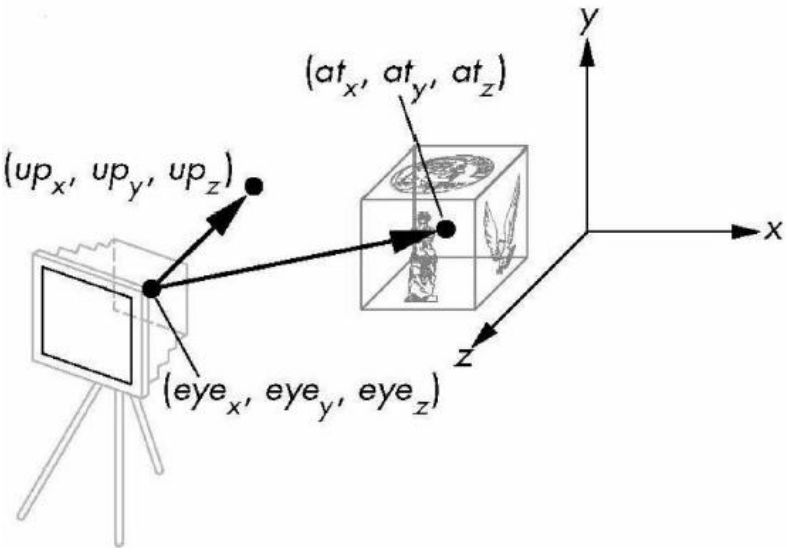
يتم تحقيق تحويل الرؤية باستخدام التابع **gluLookAt()** حيث يأخذ هذا التابع تسعة بارمترات هي :

١- الثلاثة الأولى تحدد موقع العين التي تنظر إلى الشكل .

٢- الثلاثة التالية تحدد المكان الذي تقف فيه الكاميرا .

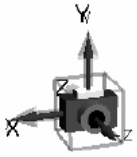
٣- الثلاثة الأخيرة تحدد من أين ينظر للشكل من فوق أو من تحت .

```
gluLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz)
```



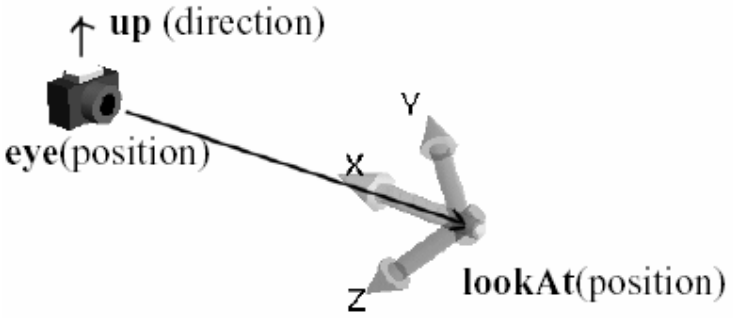
مثال ١:

بفرض لدينا الشكل التالي :



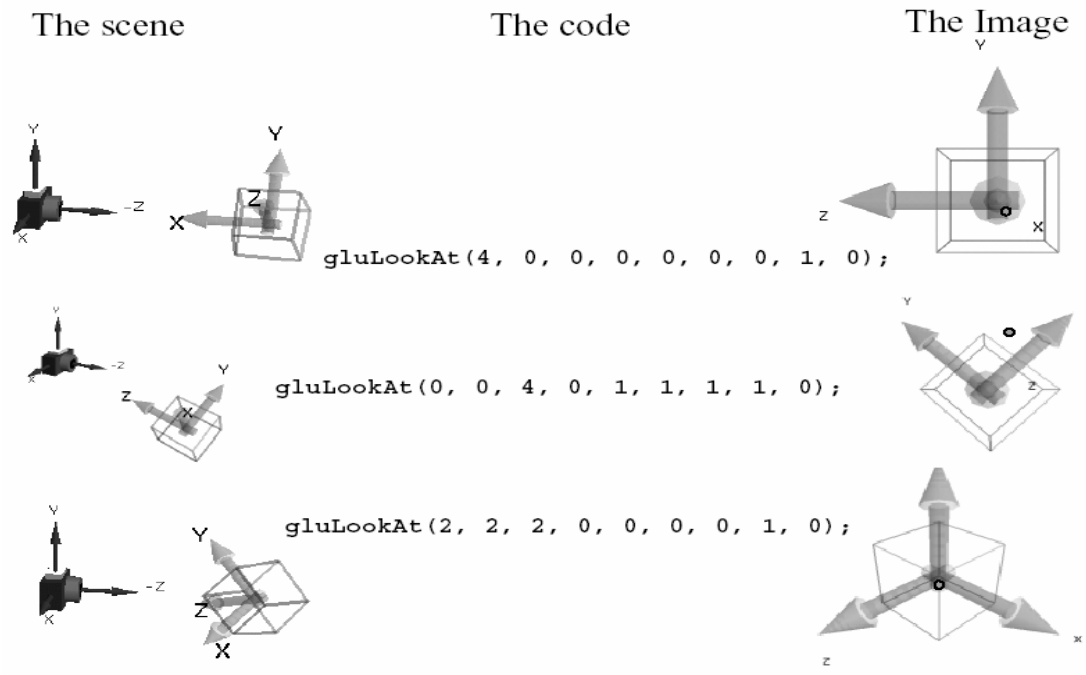

```
glLoadIdentity();
//insert camera transformation here
glutWireCube(1);
```

بحيث يحيط المكعب بكاميرا التصوير، وفي هذه الحالة لا يظهر المكعب ولجعله مرئي يجب إجراء تحويل رؤية بحيث نضع الكاميرا في مكان مناسب بالنسبة للمكعب .




```
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,
               GLdouble lookAtx, GLdouble lookAty, GLdouble lookAtz,
               GLdouble upx, GLdouble upy, GLdouble upz );
```

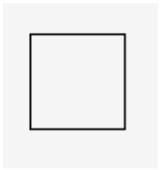
بتغيير موقع الكاميرا أو عين الناظر يتغير لدينا مظهر الشكل المعروض ولنأخذ عدة أمثلة عن المكعب .



مثال٢:

تطبيق تحويلي النمذجة والرؤية في نفس الوقت على المكعب وسنأخذ من تحويلات النمذجة تحويل التقييس .

بفرض لدينا المكعب بالشكل التالي (قبل إجراء أي تحويل) :

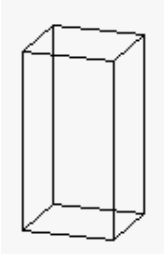


ولتطبيق تحويلي النمذجة والرؤية يجب اتباع الخطوات التالية :

- تحديد نوع التحويل المستخدم من خلال مصفوفة التحويلات `.glMatrixMode(GL_MODELVIEW);`
- تهيئة المصفوفة الواحدية بالقيم الافتراضية `.glLoadIdentity`
- استدعاء تابع تحويل الرؤية `gluLookAt` مع بارمترات مناسبة .
- استدعاء تابع تحويل النمذجة `glScalef` مع بارمترات مناسبة .
- رسم الشكل المطلوب.

```
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);

   glLoadIdentity ();             /* clear the matrix */
           /* viewing transformation  */
   gluLookAt (1.0, 1.0, 3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
   glScalef (1.0, 2.0, 1.0);      /* modeling transformation */
   glutWireCube (1.0);
   glFlush ();
}

void reshape(int w, int h)
{
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();

   glOrtho(-5.0, 5.0, -5.0, 5.0, -5.0, 5.0);

   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();         /* clear the matrix */

}
```

فيظهر لدينا الخرج التالي :

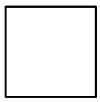


ملاحظة :

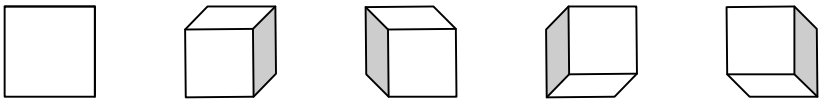
لا تقلل بشأن التابع glMatrixMode (GL_PROJECTION) حيث سنأتي على شرحه لاحقاً ، ولكن أتينا على ذكره هنا لعرض الشكل النظامي والمألوف للإجرائية reshape.

تمرين١ :

اكتب تطبيق يظهر فيه المكعب السابق بالشكل التالي:



ثم قم بتغيير قيم بارمترات التابع gluLookAt ولاحظ الأشكال المختلفة التي يأخذها هذا المكعب . استخدم لوحة المفاتيح للتحكم بقيم التابع gluLookAt .



تمرين٢ :

أكتب برنامج تعرض فيه نموذج لسيارة مثبتة على سطح قابل للدوران ، وباستخدام الفأرة والقوائم ولوحة المفاتيح تتحكم بهذه السيارة بحيث:

١. تقوم بتدوير السطح المثبت عليه السيارة باستخدام تحويلات النمذجة.
٢. تقوم بتثبيت السطح وتحريك السيارة باستخدام تحويلات الرؤية وإجراء عمليات عرض لها من جميع الجهات.

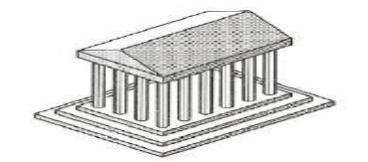
▫ تحويل الإسقاط Projection Transformations :

الإسقاط : هو تحويل مجموعة من النقاط من نظام إحداثيات بالبعد n إلى نظام إحداثيات ببعد أقل من n، وهنا سوف نحصر أنفسنا بالإسقاط من 3d إلى 2d ، حيث يستخدم الإسقاط للانتقال من جملة إحداثيات الرؤية إلى جملة إحداثيات جهاز العرض وهي 2d.

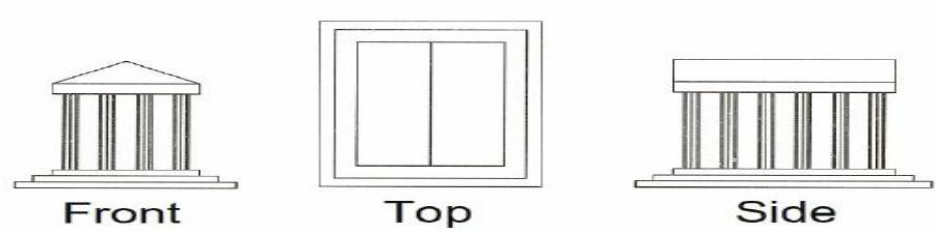
ينقسم الإسقاط إلى نوعيين أساسيين هما : **المتلاشي** perspective، **المطابق** parallel.

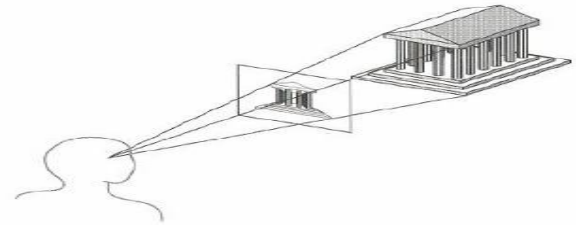
٣. **الإسقاط المتلاشي أو المنظوري perspective** : حيث أن مجموعة المستقيمات المتوازية من الممكن أن تظهر غير متوازية على مستوي الإسقاط وتميل للالتقاء في نقطة واحدة تسمى نقطة التلاشي.
٤. **الإسقاط المطابق parallel:** حيث يكون مركز الإسقاط في اللانهاية، ويكون شعاع الإسقاط موازياً لسطح الإسقاط .

بفرض لدينا النموذج التالي :



فعند الإسقاط في المستوي المتوازي Orthographic يظهر لدينا النموذج بالشكل التالي:

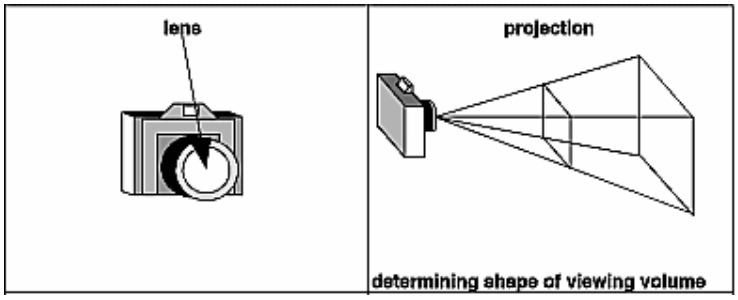




وباستخدام الإسقاط في الفراغ فإن الأجزاء الداخلية من النموذج ستظهر نوعاً ما، مما يجعله يبدو بشكل مجسّم.

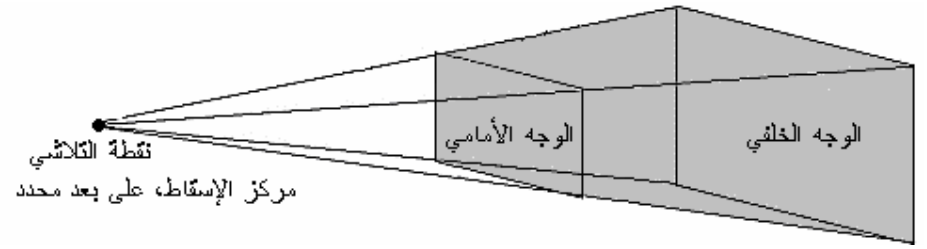
ويفيد الإسقاط في تحديد حيز الرؤية الذي نستطيع أن نرى كل ما يقع ضمنه، ولتوضيح هذا الأمر بشكل أكبر لنعود إلى تحويل الرؤية حيث استطعنا باستخدام هذا التحويل التحكم بمكان عين الناظر وتحديد الجهة التي ينظر إليها وذلك لمحاكاة ما يقوم به المصوِّر عندما يتخذ مكاناً مناسباً لالتقاط الصورة بكاميرا التصوير، ولكن

هذا ليس كل شيء، فالمصور بعد أن يحدد المكان الذي يقف به والجهة التي سيصور منها، يبدأ بالتحكم بعدسة كاميرا التصوير فيزيد من فتحة العدسة أو ينقصها وذلك لتحديد الحيز الذي يمكن للكاميرا تصويره. هذا الحيز سميناه بحيز الرؤية وهو ما يحدده لنا تحويل الإسقاط .



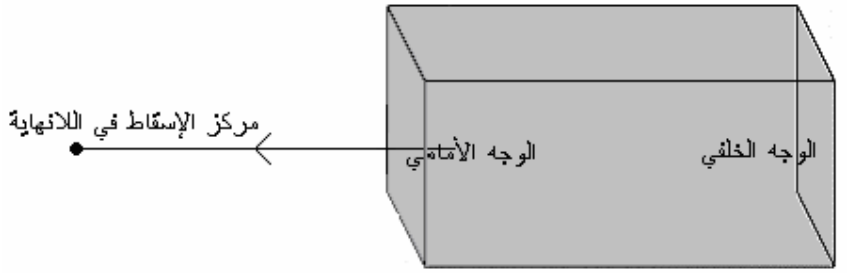
وبما أنه يوجد نوعين من الإسقاط فهذا يعني أنه يمكن تحديد نوعين من حيز الرؤية.

٣. في حالة الإسقاط المنظوري :



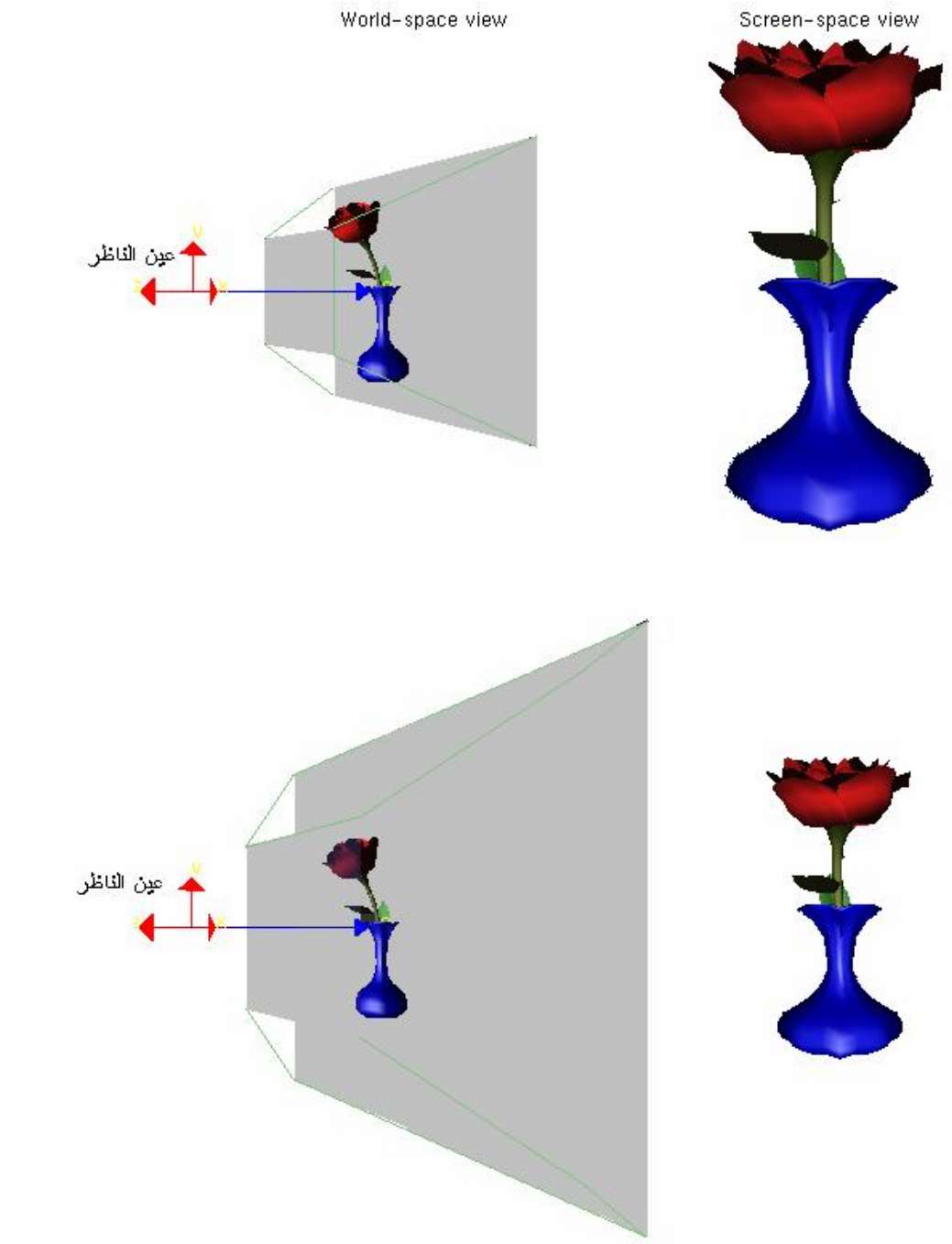
حيث يكون حيز الرؤية عبارة عن جزع الهرم المحدد بالمستويين الأمامي والخلفي.

٤. في حالة الإسقاط المتوازي:



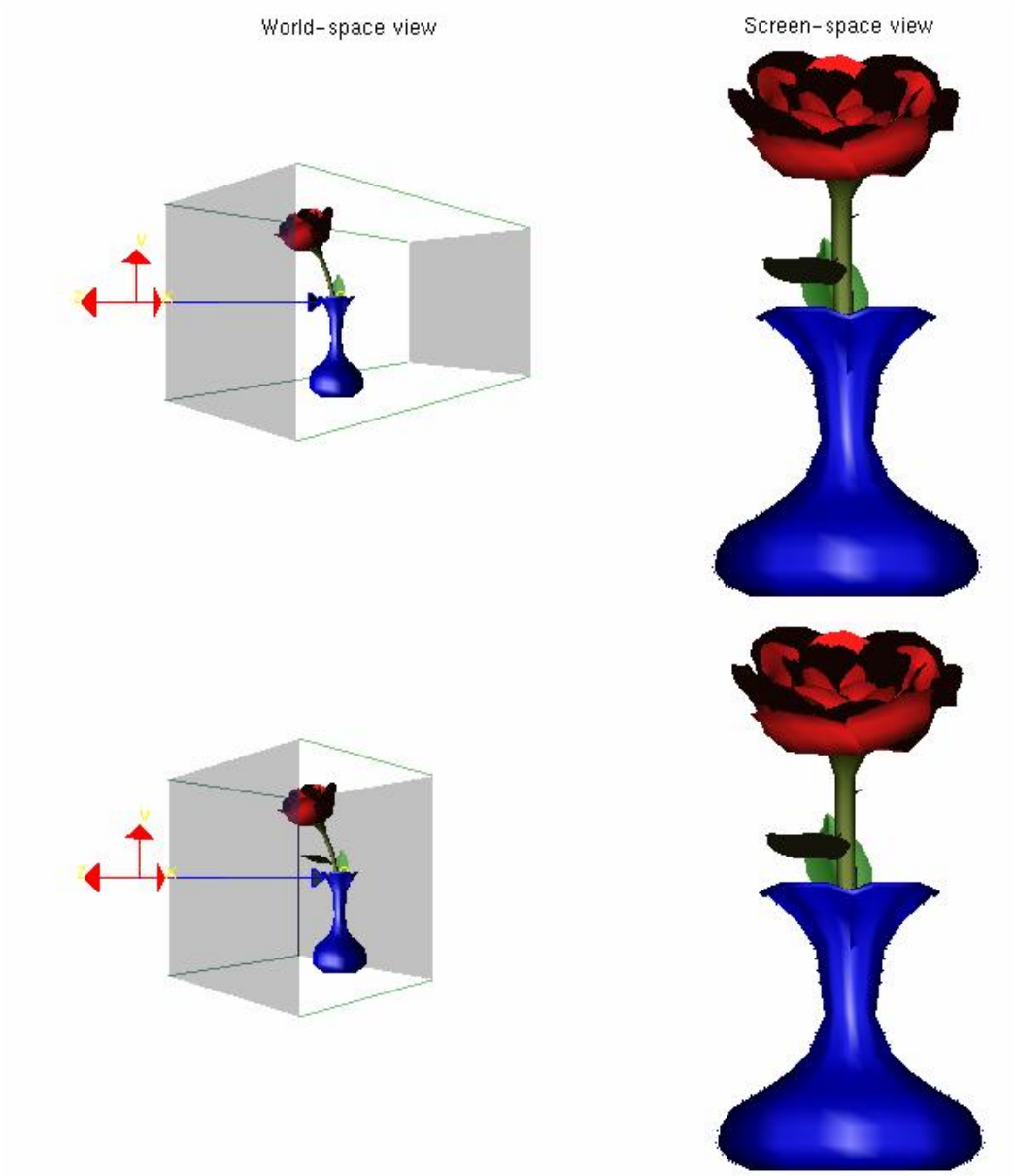
حيث يكون حيز الرؤية عبارة عن متوازي المستطيلات المحدد بالوجهين الأمامي والخلفي.

ويظهر الاختلاف بين نوعي الإسقاط السابقيين عندما نقوم بتقريب أو تبعيد كاميرا التصوير أي عندما نقوم بإجراء عملية zoom لعدسة الكاميرا ، ففي الإسقاط المنظوري أو المتلاشي يسبب إجراء عملية التقريب تكبير النموذج المصور، في حين تسبب عملية التبعيد تصغير النموذج. سنوضح ذلك بالشكل التالي:



في الإسقاط المنظوري

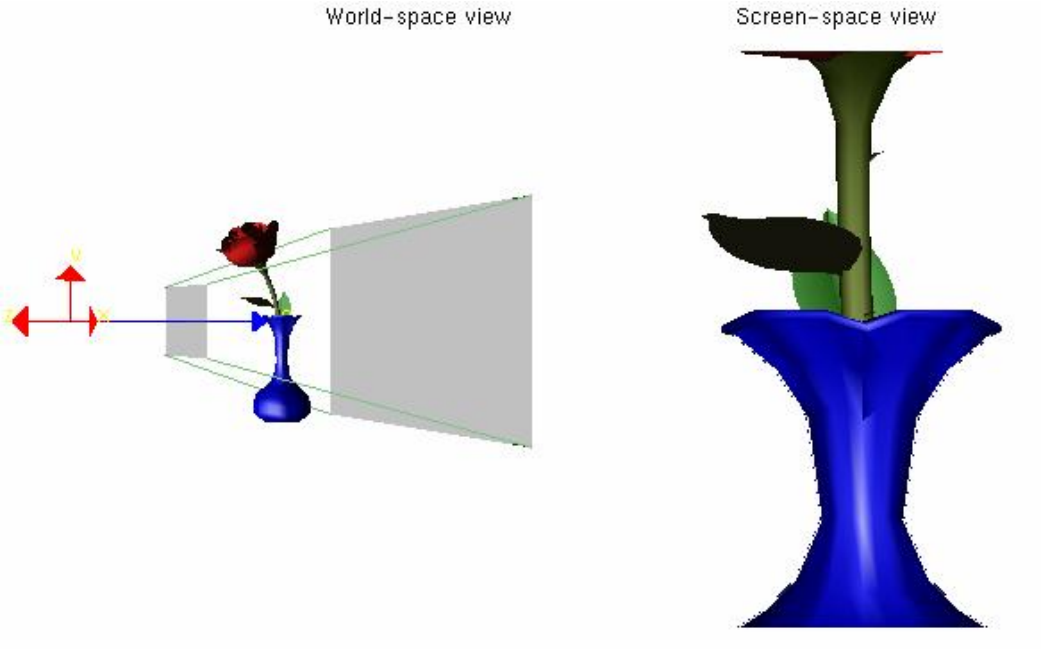
أما في الإسقاط المتوازي فإن أي عملية تقريب أو تبعيد و إجراء عملية zoom لعدسة الكاميرا لن يغير من حجم الجسم المعروض كما في الشكل التالي :



في الإسقاط المتوازي

كما يؤثر تكبير حيز الرؤية في إظهار نماذج أكثر و تصغير حيز الرؤية يسبب إخفاء بعض الكائنات التي لا تنتمي لحيز الرؤية.

وقد تسبب عملية التقريب أو التبعيد بعض الأخطاء إذا لم تخضع لقوانين معينة، حيث يمكن أن تسبب الخروج من حيز الرؤية بحيث يصبح الشكل أو جزء منه غير مرئي. كما في الشكل التالي:



لاحظ كيف ظهر جزء فقط من الشكل.

وقبل أن نتعرف على القوانين التي يجب مراعاتها للتجنب بالوقوع بمثل هذه الأخطاء، سنتعرف على التوابع التي تقدمها الـــOpenGL لتحقيق تحويل الإسقاط بنوعيه.

١- استدعاء مصفوفة التحويل الخاصة بتحويل الإسقاط:

```
glMatrixMode (GL_PROJECTION);
```

٢- تهيئتها بالقيم الافتراضية :

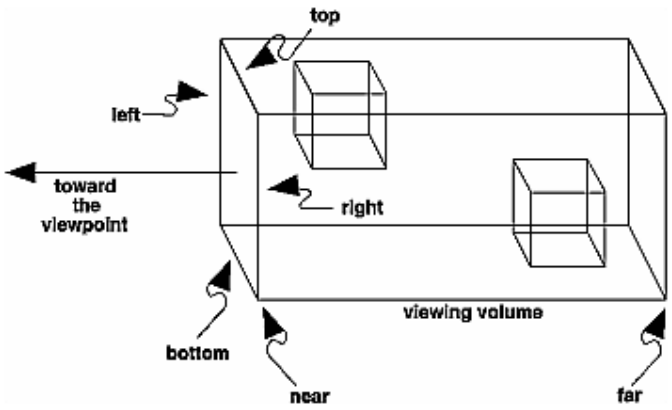
```
glLoadIdentity();
```

٣- تحديد نوع الإسقاط المستخدم :

الإسقاط في المستوي المتوازي:

عندها نستخدم التعليمة التالية:

```
glOrtho(left, right, bottom, top, Znear, Zfar);
```



ولتحديد قيم وسطاء هذه التعليمة ، يجب تحديد المعلومتين التاليتين :

١- حجم حيز الرؤية (البعد بين الوجه الأمامي والوجه الخلفي) الذي نريد أن يظهر ضمنه نموذجنا، وسنرمز له بالرمز D.

٢- تحديد إحداثيات مركز الحيز والتي نعبر عنها بالثلاثية التالية: (c.x, c.y, c.z).

عندها تكون قيم وسطاء التابع glOrtho كالتالي:

- Znear = 1.0;
- Zfar = Znear + D;
- GLdouble left = c.x - D;
- GLdouble right = c.x + D;
- GLdouble bottom c.y - D;

- GLdouble top = c.y + D;

وعندها نكتب في برنامجنا التعليمات التالية:

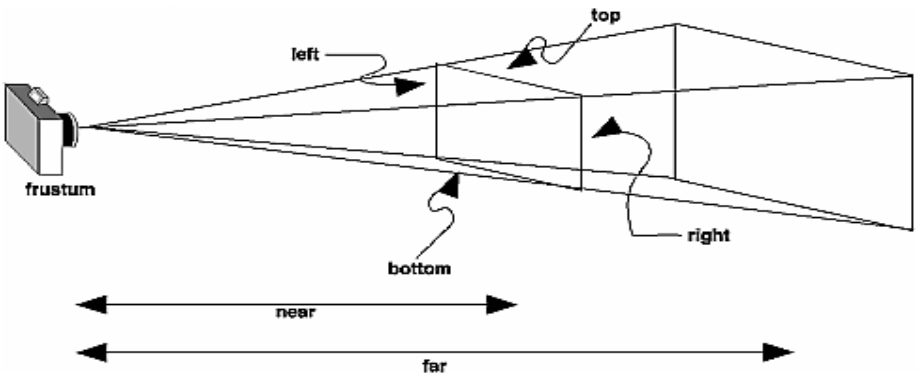
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(left, right, bottom, top, Znear, Zfar);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

الإسقاط في المستوي المتلاشي أو المنظوري:

وتقدم الـــOpenGL تعليمتين من أجل الإسقاط المتلاشي هما:

- **التعليمة glFrustum** :

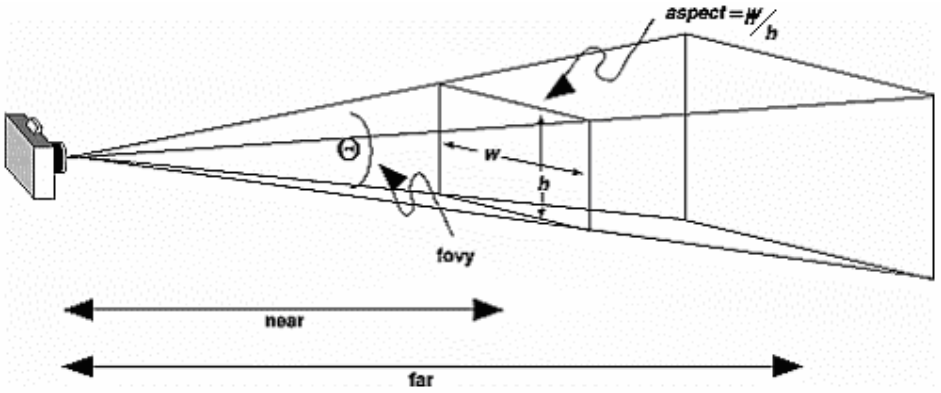
```
glFrustum (left,right,bottom,top,Znear,Zfar);
```



وتحدد بارمترات هذه التعليمة تماماً كما في التعليمة glOrtho.

- **التعليمة gluPerspective**:

```
gluPerspective (fov, aspect, Znear, Zfar);
```



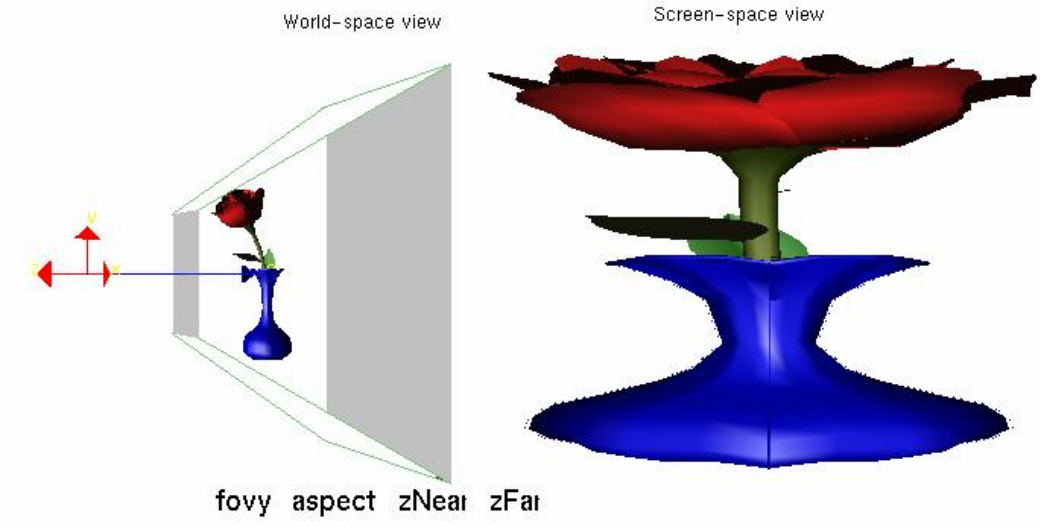
ويمثل الوسيط fov فتحة العدسة أو زاوية الإظهار، ويمثل الوسيط aspect نسبة عرض النافذة على طولها، فإذا كانت النافذة بشكل مربع عندها تكون قيمة الـ aspect=1، ولكن عندما تكون غير ذلك عندها نستخدم المعادلات التالية لحساب قيم الـ bottom، top، left، right:

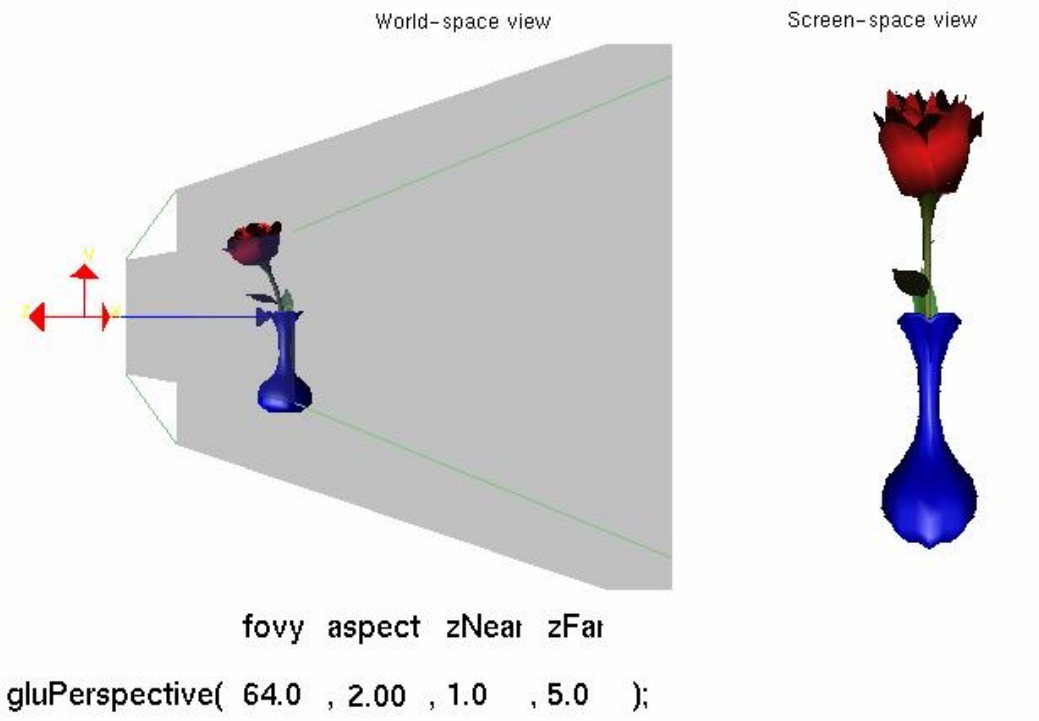
```
GLdouble aspect = (GLdouble) windowWidth /windowHeight;
if ( aspect < 1.0 ) { // window taller than wide
   bottom /= aspect;
   top /= aspect;
} else {
   left *= aspect;
   right *= aspect;
}
```

إن التعليمتين **glFrustum**، **gluPerspective** تقومان بنفس العمل ولكن تعتبر التعليمة glFrustum تخيلية وتعتمد على القيم التجريبية، في حين تعتبر التعليمة gluPerspective أكثر استخداماً وواقعية .

في المثال السابق الذي عرضنا فيه نموذج الوردة استخدمنا تعليمة gluPerspective في الإسقاط المتلاشي ومن أجل تكبير أو تصغير حيز الرؤية كنا نزيد من قيمة الوسيط fov أو ننقصه .
وبالنسبة لنموذج الوردة في الإسقاط المتوازي فإننا تحكمنا بقيمة Zfar بالزيادة أو النقصان.

أما لمعرفة الأثر الناتج عن تغيير قيم الوسيط aspect فسنعرض المثال التالي:

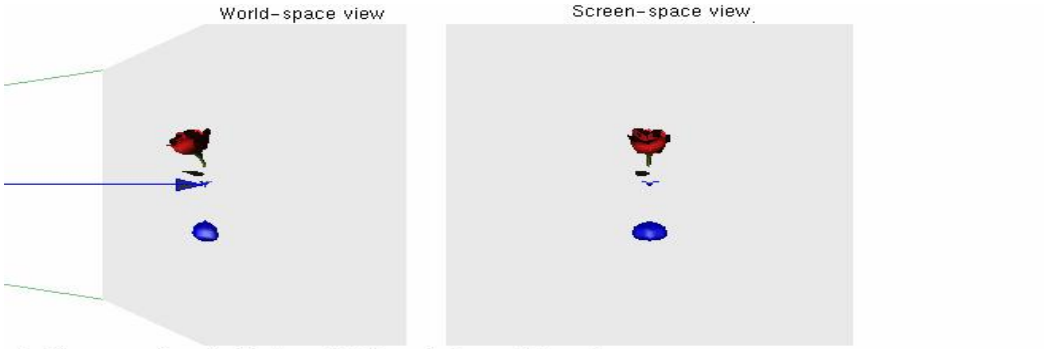




وبالنسبة لتحويل الرؤية، تستدعى التعليمة gluLookAt بالشكل التالي:

```
gluLookAt (0.0, 0.0, D/2,c.x, c.y, c.z,0.0, 1.0, 0.0);
```

حيث أن الشعاع z المتعلق بموقع العين يجب أن ينتمي لمجال حيز الرؤية بحيث يكون أكبر مـــن Znear، وأصغر من Zfar . وإلا فإن النموذج سيصبح غير مرئي.




```
gluPerspective( 60.0  , 1.00  , 0.1   , 5.0   );
gluLookAt( 0.00  , 0.00  , 5.00  ,  0.00  , 0.00  , 0.00  , 0.00  , 1.00  , 0.00  );
```

لاحظ كيف عندما بدأت zEye تتساوى مع zFar بدأ النموذج بالاختفاء.

والآن بفرض أن المستخدم يريد أن يقوم بعملية التكبير أو التصغير بواسطة متحول وليكن zoomFactor ، عندها تستدعى التعليمات السابقة كالتالي :

```
glOrtho (left*zoomFactor, right*zoomFactor,
         bottom*zoomFactor, top*zoomFactor,
          zNear, zFar);

glFrustum(left*zoomFactor, right*zoomFactor,
           bottom*zoomFactor, top*zoomFactor,
            zNear, zFar);
gluPerspective (50.0*zoomFactor,
                    (float)width/(float)height,
                     zNear, zFar);
```

مع ملاحظة أن أي خطأ بالقيم الممرة قد يتسبب بعدم ظهور النموذج.

مثال١:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

gluPerspective(50.0, 1.0, 3.0, 7.0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

gluLookAt(0.0, 0.0, 5.0,0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
```

لاحظ كيف يعمل التحويلين الإسقاط والرؤية مع بعضهما.

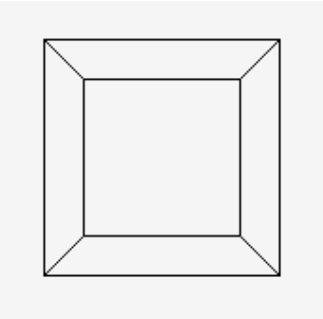
لدينا في هذا المثال زاوية الرؤية تساوي ٥٠ درجة، الـ aspect تساوي واحـد، Znear=3، Zfar=7، عندها يكون حجم حيز الرؤية يساوي ٤. لاحظ الآن أن مكان عين الناظر تقع في (0.0,0.0,5.0)، وأن اتجاه النظر نحو المركز (0.0,0.0,0.0) . هذا يعني أن عين الناظر تبعد عن النقطة التي ننظر إليهـا بمقدار ٥ وبالتالي فهي تقع ضمن حيز الرؤية الذي يبدأ عند Znear=3 وينتهي عنـد Zfar=7. ولكـن إذا وضعنا عين الناظر في الموقع (0.0,0.0,1.0)أو (0.0,0.0,10.0) عندها لن يظهر أي جزء مـن النموذج لأن موقع العين لم تعد داخل حيز الرؤية.

مثال٢:

بفرض لدينا مكعب، نريد تطبيق تحويل الإسقاط عليه :

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

GLdouble D;

void init(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0;(
   glShadeModel (GL_FLAT;(
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   /* viewing transformation  */
   gluLookAt (0.0, 0.0, D/2, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
   glutWireCube (1.0);
   glFlush ();
}
void reshape(int w, int h)
{
   D = 5;

  GLdouble aspect = (GLdouble) w / h  ;
  GLdouble Znear = 1.0;
  GLdouble Zfar = Znear + D;

  GLdouble cx = 0;
  GLdouble cy = 0;

  GLdouble left = cx - D;
  GLdouble right = cx + D;
  GLdouble bottom = cy - D;
  GLdouble top = cy + D;

  if ( aspect < 1.0 ) { // window taller than wide

       bottom /= aspect;
       top /= aspect;

  } else {
   left *= aspect;
   right *= aspect;
  }
```

عندها سيكون الخرج كالتالي :

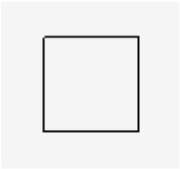


وباستخدام التعليمة glFrustum نفس قيم الوسطاء سنحصل على الخرج التالي:

```
glFrustum (left, right, bottom, top, Znear, Zfar);
```

لاحظ الاختلاف بالنتائج ، وهذا ما أشرنا إليه بأن الـ gluPerspective أقرب للواقع وأن glFrustum تعتمد على القيم التجريبية.

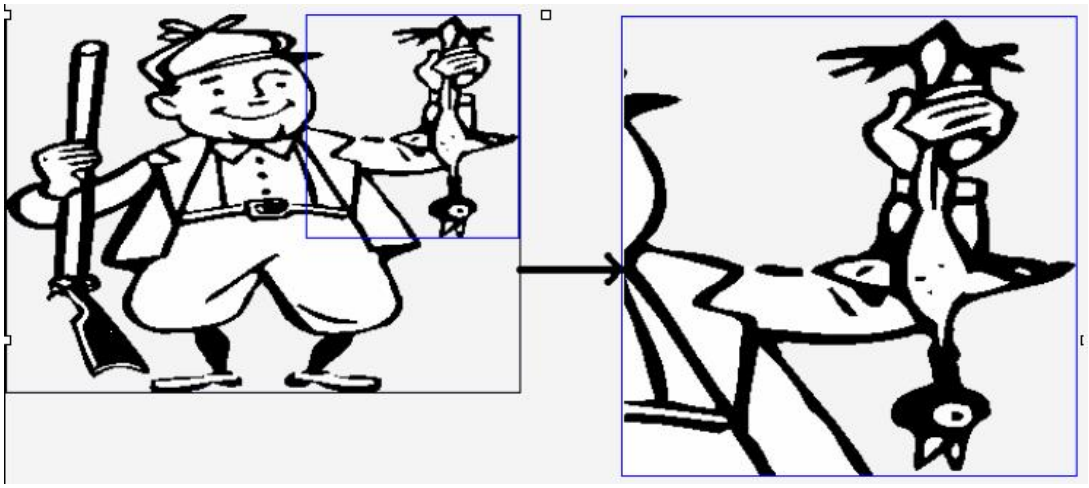
وإذا استخدمنا التعليمة glOrtho فسنحصل على النتيجة التالية:



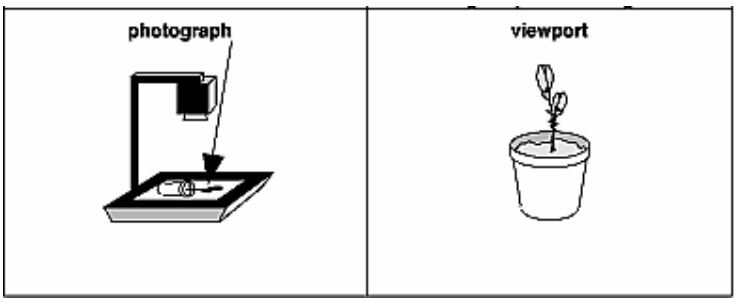
وهذا ما سبق وذكرناه أيضاً بأن الأجزاء الداخلية من الشكل في الإسقاط المتوازي لن تظهر وسنرى فقط الوجه الأمامي .

▫ تحويل نافذة الرؤية ViewPort Transformation :

تعريف نافذة الرؤية :عبارة عن مستطيل يُــقطَع كل إظهار تتجاوز إحداثياته حدود هذا المستطيل ،ويطلــق عليه اسم بوابة العرض أيضاً . ولنأخذ الشكل التالي كمثال للتوضيح :



لاحظ من الشكل المستطيل الملون باللون الأزرق ، يمثل هذا المستطيل نافذة الرؤية ، بمعنى أنه لن يظهر من الشكل إلا هذا الجزء المحدد بالمستطيل .ولفهم نافذة الرؤية بشكل أكبر يمكن الرجــوع لكــاميرا التــصوير وتشبيهها بالمربع الذي يظهر ضمن العدسة حيث كلنا يعلم أن الأجزاء التي توضع ضمن هذا المربع هي فقط التي ستظهر ضمن الصورة .

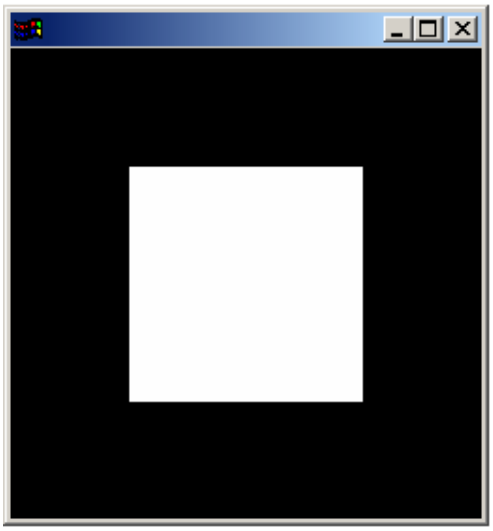


الهدف من هذا التحويل :

استطعنا سابقاً أن نظهر العديد من الأشكال دون الحاجة إلى استدعاء نافذة الرؤية ، وإذا كان الهدف الأساسي هو تحديد الأجزاء التي ستظهر فقط ، يمكن لأحدنا أن يفكر بأن هذا الموضوع يمكن التغلب عليه بعدم رسم هذه الأجزاء في الأساس ، ولكن في الواقع إن الهدف من استخدام نافذة الرؤية تتمثل بالنقاط التالية :

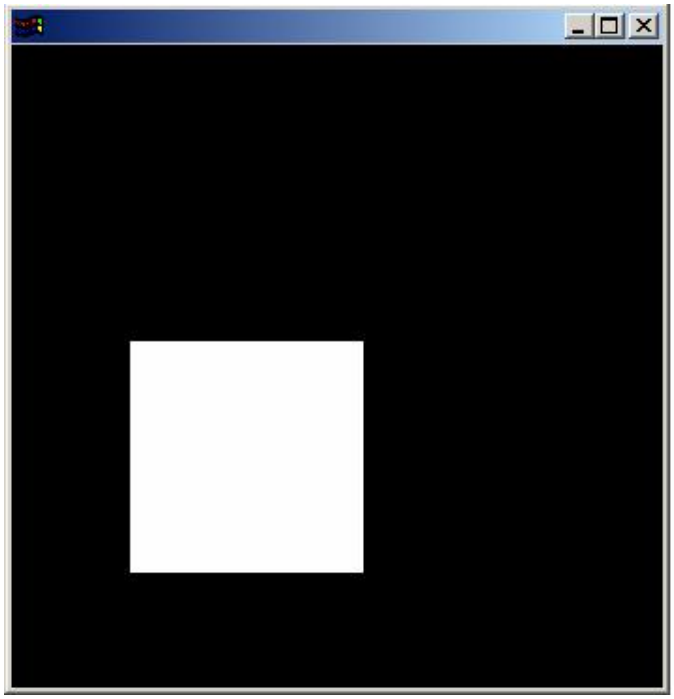
١. المحافظة على مكان ظهور الشكل على شاشة العرض البياني مهما تغيرت أبعاد النافذة الرئيسية مع المحافظة على الأبعاد النسبية له، فمثلاً لو اعتبرنا أن حجم النافذة الرئيسية هو (w=250, h=250)،

وأن الشكل الذي نريد إظهاره هو مضلع إحداثياته هي (x1=25, y1=25, x2=75, y2=75)
فيظهر الشكل كالتالي :



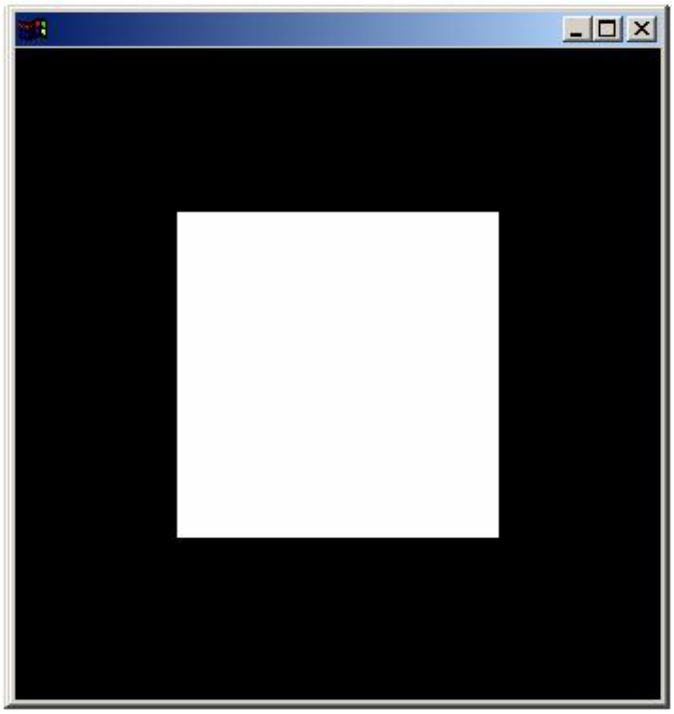
الشكل a

ولكن إذا قمنا بتكبير النافذة باستخدام زر التكبير عندها ستتغير إحداثيات النافذة ولكن إحداثيات المضلع وموقعه الأصلي في النافذة لن يتغير وسيظهر عندها بالشكل التالي :



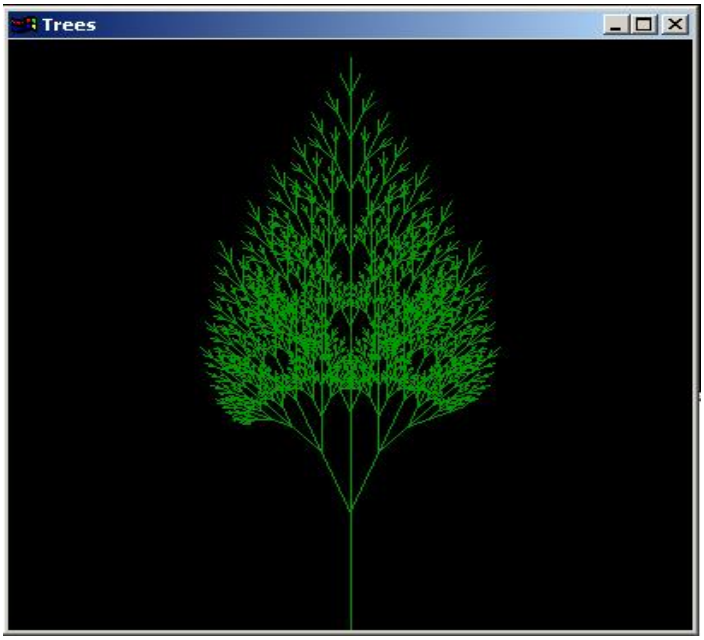
الشكل b

وبالمقارنة بين الشكلين a, b نجد أن المضلع في الشكل a كان يظهر في منتصف النافذة و في الشكل b أصبح بعيداً عن المنتصف وأقرب إلى زاوية النافذة وكذلك فإن حجمه بدا وكأنه تغيَّر مع أن موقعه و إحداثياته لم يطرأ عليهما أي تغيير ونفسر هذا الأمر بسبب عدم وجود ترابط نسبي بين حجم النافذة وحجم الشكل المعروض فعندما كبر حجم النافذة ولم يترافق هذا بكبر في حجم المضلع بدا الشكل وكأنه قد تغير مكانه وحجمه ، وقد يسبب هذا في الأشكال والنماذج الأكثر تعقيداً تشوها في الصورة ، و تحل هذه المشكلة باستدعاء نافذة الرؤية التي توجد علاقة نسبيَّة بين حجم النافذة وحجم النموذج المعروض وتحافظ على موقع الأشكال التي تظهر في النافذة سواءً أكانت هذه الأشكال تظهر في منتصف النافذة أو في أطرافها أو في أي مكان آخر ومن أجل المثال السابق إذا استدعينا نافذة الرؤية في إجرائية الـ Reshape، فسيظهر الشكل بعد تكبير النافذة كالتالي :

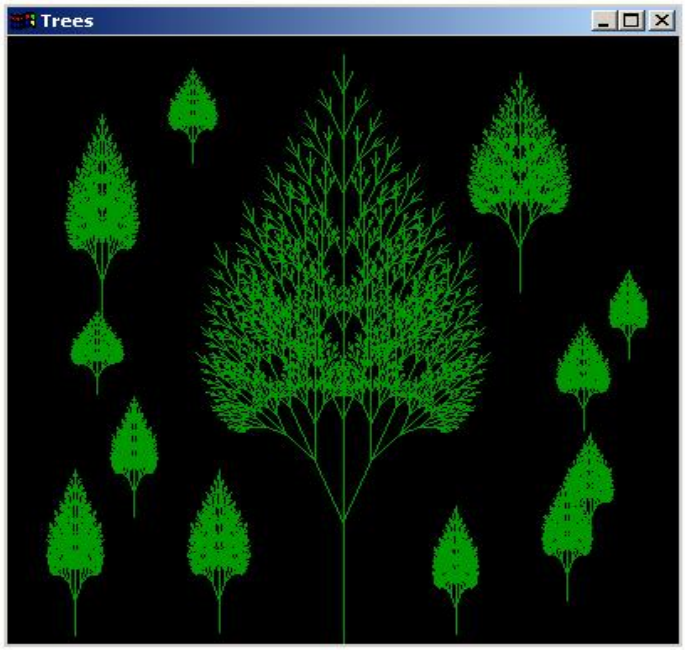


لاحظ كيف بقي الشكل في منتصف النافذة وكيف تغيَّرت أبعاده بما يتناسب مع تغُّير أبعاد النافذة

٢. إظهار أكثر من نسخة للنموذج المعروض في النافذة الرئيسية وبأحجام مختلفة :قد يحتوي النموذج المعروض على أشكال متشابهة ولكنها تختلف بحجمها فقط ،وإذا فكرنا بكتابة إجرائية عرض ترسم كل من هذه الأشكال على حدا فإن حجم البرمجة سيكون كبير جداً ومرهق للمبرمج وقد حُلَّت هذه المشكلة بإنشاء أكثر من نافذة رؤية ضمن النافذة الرئيسية وبتغيير أحجام هذه النوافذ يتغير حجم الشكل المعروض ضمنها . ولنأخذ على سبيل المثال تطبيق يعرض شجرة واحدة .

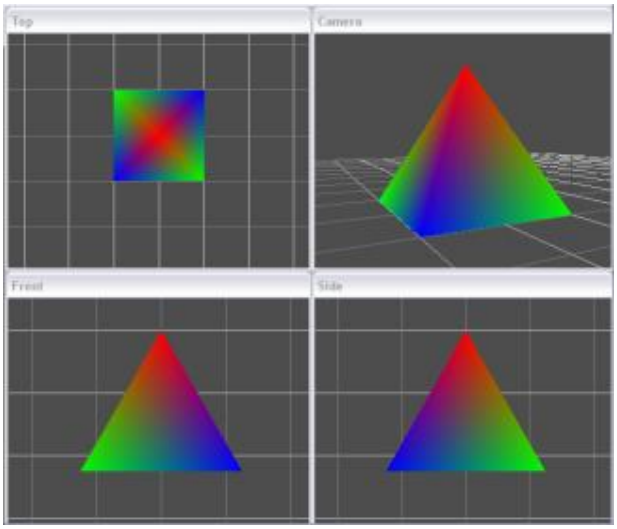


وإذا أردنا تحويل هذا النموذج لعرض حديقة من الأشجار بحيث نظهر في الحديقة أشجار كبيـــرة وأخـــرى صغيرة فلن نكون مضطرين إلى كتابة كود من أجل إظهار كل تلك الأشجار وإنما فقط ننشئ عدة نوافذ رؤية نظهر ضمنها نفس الشكل باستدعائه فقط وسنحصل على حديقة الأشجار التالية .



مثال ٢:

إظهار النموذج في عدة نوافذ رؤية من عدة جوانب.

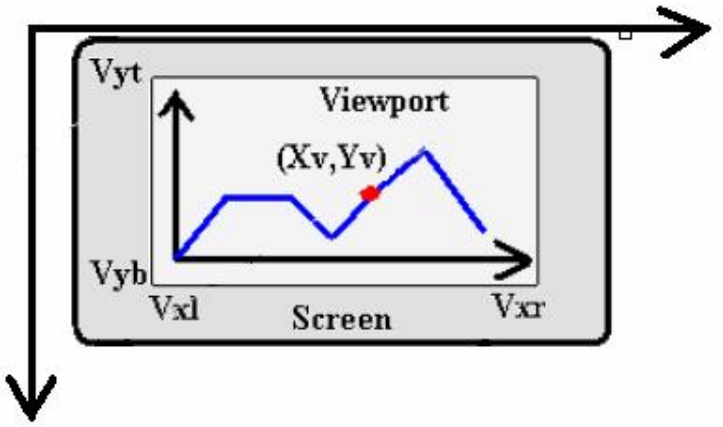


٣. كما يمكن الاستفادة أحياناً من نافذة الرؤية بتكبير أو تصغير النموذج المعروض وذلك بالتحكم بأبعاد النافذة وحجمها، وكذلك يمكن تغيير مكان النافذة أثناء تشغيل التطبيق بالتحكم بإحداثيات موقع بداية النافذة، وهذا قد يفيدنا أيضاً بتحريك النموذج المعروض .

ولإنشاء نافذة الرؤية في الـ OpenGL هناك تابع يدعى **glViewport** و بارامتراته هي الإحداثيي x,y اللذان يمثلان مبدأ الإحداثيات للنافذة و h,w اللذان يمثلان عرض وارتفاع النافذة .

```
glViewport (x,y,(GLsizei) w,(GLsizei) h);
```

وتستدعى هذه التعليمة في إجرائية الـ Reshape .و تنشأ نافذة الرؤية ضمن النافذة الأساسية التي تمثل شاشة العرض البياني .وتتشابه هاتين النافذتين إلى حد كبير إلا أنه يوجد فرق أساسي بينهما مرتبط بمبدأ الإحداثيات لكل منهما ، حيث أن مبدأ إحداثيات النافذة الأساسية في الزاوية العليا اليسارية من الشاشة ،ومحور السينات موجه من اليسار إلى اليمين ، ومحور العينات موجه من الأعلى إلى الأسفل ،أما مبدأ إحداثيات نافذة الرؤية في الزاوية السفلى اليسارية من الشاشة ، ومحور السينات موجه من اليسار نحو اليمين ومحور العينات موجه من الأسفل نحو الأعلى .

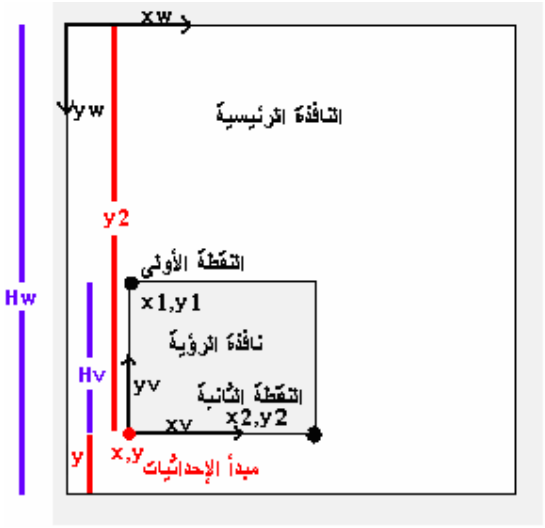


كيفية تحديد إحداثيات نافذة الرؤية :

ذكرنا أن مبدأ الإحداثيات لنافذة الرؤية في الزاوية اليسارية السفلى و لإنشاء نافذة الرؤية يجب معرفة إحداثيي المبدأ (x,y)ومن ثم حساب طول وعرض هذه النافذة حيث :

```
W = x2-x1
```

```
H = y2-y1
```



حساب إحداثيي المبدأ :

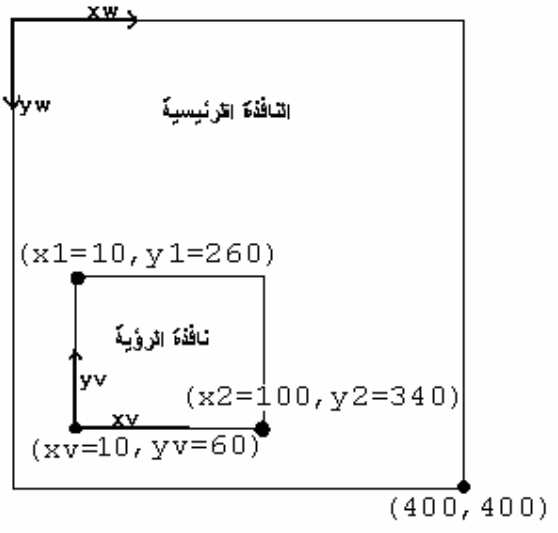
من الشكل السابق نلاحظ أن الإحداثي x =x1 ، أما الإحداثي y فيحسب كالتالي :

```
Y = Hw -y2
```

وذلك لأن y تمثل البعد عن الحافة السفلى للنافذة الرئيسية .

مثال:

بفرض أن أبعاد الشاشة الرئيسية السابقة(400,400)، وبفرض أن الإحداثيي (x1=10,y1=260)، (x2=100,y2=340) . عنـــــدها يكـــــون إحـــــداثيي المبـــــدأ لنافـــــذة الرؤيـــــة (xv= 10,yv=60)بالنسبة للنافذة الرئيسية.



قلنا في الفقرة السابقة أن إحدى الفوائد التي نحصل عليها من نافذة الرؤية هو إظهار أكثر من نـــسخة مـــن الشكل أو النموذج في النافذة الرئيسية الواحدة وأخذنا على سبيل المثال حديقة الأشجار ، وفي الواقع إن عملية إنشاء مجموعة النوافذ هذه وتشكيل حديقة الأشجار غالباً ما تكون بتدخل من المـــستخدم وذلـــك باســـتخدام الطرفيات المتاحة له مثل لوحة المفاتيح والفأرة . حيث يستعين المستخدم بالفأرة لتحديد مكان الشجرة وحجمها فيتم إنشاء نافذة رؤية في المكان والحجم الذي حدده المستخدم . و يتحدد مكان نافذة الرؤية أو الشجرة باختيار نقطة في الزاوية العليا اليسارية من الشاشة، واختيار نقطة ثانية في الزاوية السفلى اليمينية من الشاشة وتطبيق طريقة حساب مبدأ الإحداثيات السابقة.

مقارنة بين نافذة الرؤية والنوافذ الجزئية:

النوافذ الجزئية :

يتم إنشاء نوافذ جزئية ضمن النافذة الأساسية ، ويتم التعامل مع كل نافذة بشكل مستقل عن بقية النوافذ ، ولكل نافذة جزئية إجرائيات تهيئة وإعادة رسم و إجرائية عرض وغيرهـــامن الإجرائيـــات المتعلقـــة بالأحـــداث (كإجرائية الفأرة ، لوحة المفاتيح، القوائم) الخاصة بها.

نقاط التشابه:

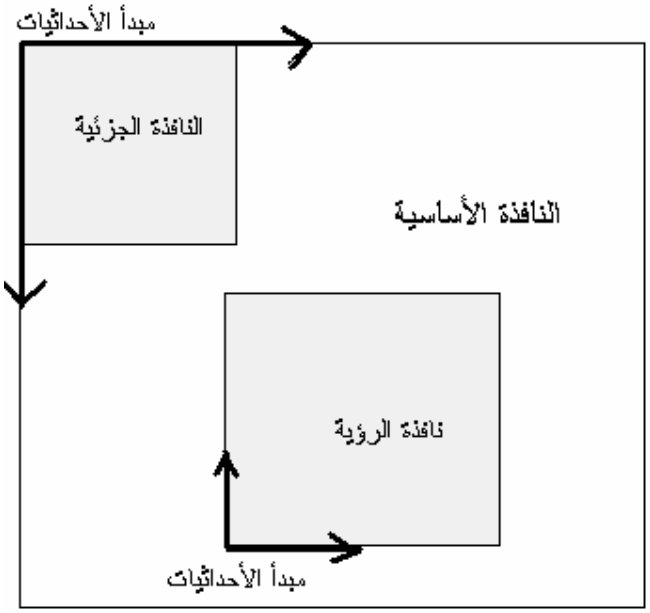
تشبه نافذة الرؤية النوافذ الجزئية بـــ:

١. يمكن إنشاء عدة نوافذ رؤية وعدة نوافذ جزئية في التطبيق،
٢. كما يمكن التحكم بحجم النوافذ الجزئية تماماً كنوافذ الرؤية.
٣. وتخضع النوافذ الجزئية لتحويلات النمذجة كنوافذ الرؤية

نقاط الاختلاف:

تختلف نافذة الرؤية عن النوافذ الجزئية بعدة نقاط نذكر منها:

١. مبدأ الإحداثيات : حيث تتشابه النوافذ الجزئية بمبدأ إحداثياتها مع النافذة الرئيسية حيـــث يعتبـــر مبـــدأ الإحداثيات في الزاوية العليا اليسارية ، في حين مبدأ إحداثيات نافذة الرؤية في الزاوية السفلى اليسارية .



٢. النماذج الرسومية : في نافذة الرؤية يتكرر نفس الشكل المعروض في النافذة الأساسية بأحجام مختلفـــة ولكن لا يمكن إظهار أشكل جديدة في نافذة الرؤية على العكس من النوافذ الجزئية والتي تتمتع باستقلاليه بنماذجها وأشكالها التي ترسمها حيث يمكن إظهار أشكال مختلفة وغير مرسومة في النافذة الأساسية .

٣. ترتبط النوافذ الجزئية مع النافذة الأساسية بعلاقة أب- ابن. حيث تعتبر النوافذ الجزئية الموجودة في نفس المستوى أبناء للنافذة الأساسية، في حين لاتوجد هكذا علاقة بين اوافذ الرؤية والنافذة الرئيسية.

٤. يمكن إنشاء عدة نوافذ جزئية ضمن نافذة جزئية ما ، كما يمكن إنشاء عدة نوافذ رؤية ضمن النافذة الجزئية أيضاً.

بعض التعليمات المتعلقة بالنوافذ الجزئية :

١. إنشاء النوافذ الجزئية: يمكن إنشاء عدد غير محدد من النوافذ الجزئية، وتعيد هذه الإجرائية رقم النافذة الجزئية المولدة و نستفيد من هذا الرقم كثيراً في التطبيقات ، أما بارمتراتها فهي :

رقم النافذة الأب التي أنشئت فيها النافذة الجزئية. وباقي البارمترات فهي تخص أبعاد النافذة الجزئية المراد إنشاؤها.

```
int glutCreateSubWindow(int win,int x, int y, int width, int height);
```

٢. تفعيل النافذة الجزئية : حتى نتمكن من التعامل مع النافذة الجزئية كإنشاء قوائم أو إظهار نموذج رسومي و تفعيل أحداث الفأرة ولوحة المفاتيح يجب تفعيل النافذة المراد التعامل معها أولاً ونستخدم لذلك التعليمة التالية:

```
void glutSetWindow(int win);
```

بارمتر هذه التعليمة هو رقم النافذة الجزئية (الذي تولد عند الإنشاء) المراد تفعيلها.

٣. الحصول على النافذة الجزئية المفعّلة : ولمعرفة النافذة الجزئية الحالية يمكن استخدام التعليمة:

```
int glutGetWindow(void);
```

حيث تعيد رقم النافذة الجزئية المفعّلة.

٤. هدم النافذة الجزئية: لتخلص من النافذة الجزئية يمكن هدمها باستخدام التعليمة التالية:

```
void glutDestroyWindow(int win);
```

التي نمرر لها رقم النافذة المراد هدمها.

تمرين١ :

اكتب تطبيق رسومي تظهر فيه الشجرة بحسب الشكل السابق، ومن ثم حول هذا التطبيق ليكون تفاعلي بحيث يمكن للمستخدم تحديد نافذة الرؤية وإظهار شجرة بداخلها باستخدام أحداث الماوس.

تمرين٢ :

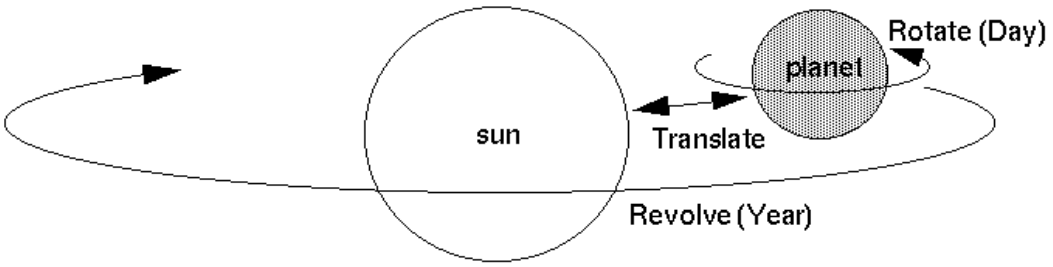
اكتب تطبيق يرسم فيه مصور الجمهورية العربية السورية مع تسمية الحدود وبعض المدن الرئيسية. تؤخذ المعلومات المتعلقة بالحدود من ملف جاهز مسبقاً كما يظهر فيه:

- دليل المصور أسفل ويسار النافذة الرئيسية .
- إظهار نسخ صغيرة للمصور في الزوايا الباقية للنافذة الرئيسية (باستخدام subWindow)، كل نسخة تحوي معلومات إحصائية معينة (أمطار ، سكان، كثافة، مساحة) . تؤخذ المعلومات من ملف جاهز مسبقاً ويتم إظهار المعلومة بشكل بياني مناسب.
- عند الضغط على إحدى النسخ الموجودة في زوايا النافذة يتم تحريكها باتجاه وسط النافذة مع تكبير مناسب لتحل محل المصور الأصلي.
- رسم مخططات بيانية للمعلومات الإحصائية (Histograms).
- رسم علم الجمهورية على أية بوابة رؤية يتم تحديدها عن طريق الفأرة .

ملاحظة: يتم تنفيذ المطلوب باستخدام لوحة المفاتيح والفأرة والقوائم.

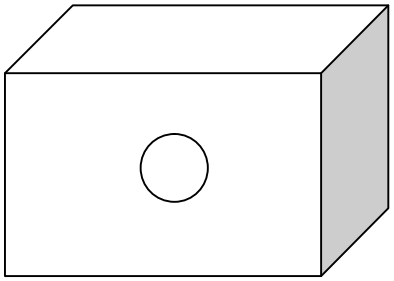
تمرين٣:

اكتب تطبيق يظهر حركة الأرض حول الشمس، مع أخذ حركة الأرض حول نفسها بعين الاعتبار مطبقاً كل التحويلات اللازمة التي تعلمتها سابقاً .



تمرين٤:

بفرض لدينا كرة موضوعة داخل مكعب وبإعطاء هذه الكرة سرعة ابتدائية تبدأ بالحركة داخل المكعب، وعند اصطدامها بحواف المكعب تتخامد سرعة هذه الكرة حتى تتوقف. اكتب تطبيق يظهر حركة الكرة و تخامدها مستفيداً من قانون الطاقة الحركية ومن التحويلات التي تعلمتها .



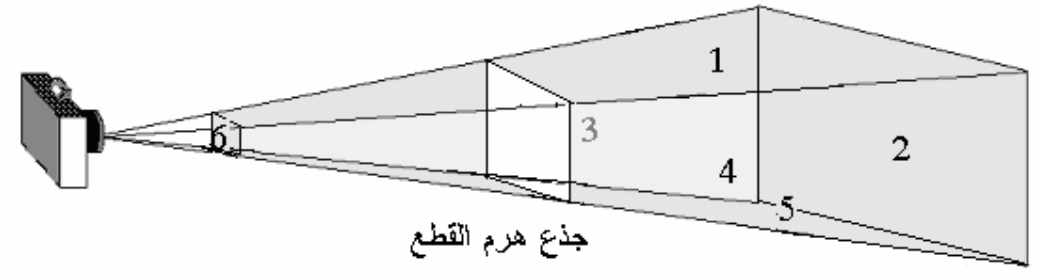
▫ تحويل القطع Clipping Transformation :

تعريف القطع :

وهو اقتطاع لجزء من النموذج المرسوم يقع خارج حدود شاشة الإظهار أو خارج حواف نافذة معينة، تدعى هذه الحواف بحدود القص . وتنجز عملية القص هذه وفق مفهوم الإخفاء لإزالة كل شيء خارج مجال القص . فمثلاً في مشهد ما إذا غيرنا زاوية الرؤية (تغيير مكان الكاميرا) ستتغير الأجزاء التي ستظهر من النموذج وكذلك الأجزاء التي ستختفي منه .

جذع هرم القطع:

هو مخروط ناقص له ستة سطوح يتم القطع وفقاً لها. وهذه السطوح تكون عادةً عمودية على المحاور الإحداثية الخاصة بمجال إحداثيات العين.

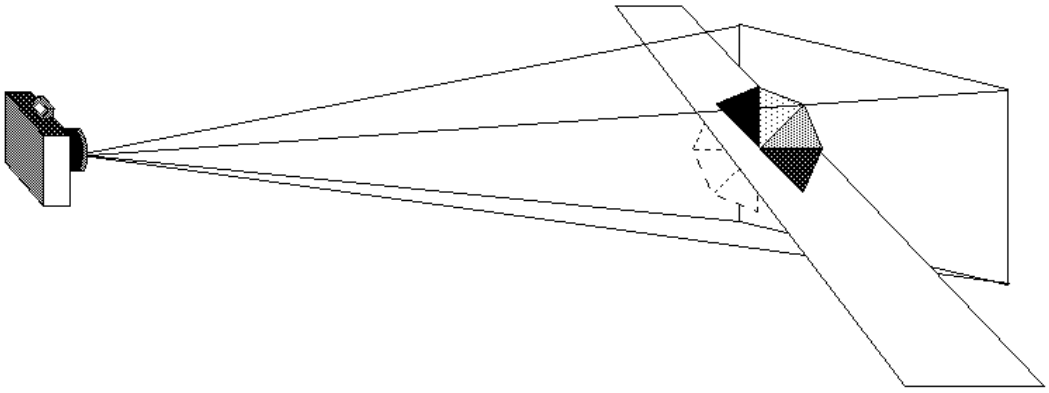


جذع هرم القطع

كل ما يقع ضمن جذع الهرم يعد مرئي، وكل مايقع خارج هذا المجال غير مرئي.

استطاعت الـــ OpenGL أن تحقق عمليات القطع والإخفاء وفق هذه السطوح الستة ، **بالإضافة إلى إمكانية تعريف سطوح قطع أخرى تعرف بالمعادلة التالية:**

$$Ax+By+Cz+D = 0.$$



تخضع هذه المعادلة لتحويل يجعلها ضمن مجال هرم الرؤية، وهذا التحويل يتم باستخدام معكوس مصفوفة التحويلات modelview matrix ، وكل نقطة تحقق المتراجحة التالية:

(A B C D)M-1 (xe ye ze we)T >= 0 تقع ضمن مجال الرؤية، حيث xe ye ze we نقاط إحداثيات العين (الرؤية) .

تابع القطع في الـــ OpenGL:

تحقق الـــOpenGl عملية القص والإخفاء باستخدام تابع القص الذي يأخذ الشكل التالي :

```
void glClipPlane(GLenum plane, const GLdouble *equation);
```

بارمتراته :

Plane : تحديد السطح الذي سيتم اقتطاعه ويعرف على شكل ثابت يأخذ القيمة **GL_CLIP_PLANE**i حيث i هو عدد صحيح يتحدد من خلاله رقم السطح الذي سيُقتطَع .

equation : عبارة عن شعاع من أربع معاملات

$$Ax+By+Cz+D = 0$$

يمثل معادلة سطح القطع الذي يحدده المستخدم .

بالإضافة إلى هذا التابع فإننا بحاجة إلى استخدام تابع لتفعيل عملية القطع وهذا التابع هو :

```
glEnable(GL_CLIP_PLANEi);
```

و بالمقابل يمكن عدم تفعيل القطع بالتابع

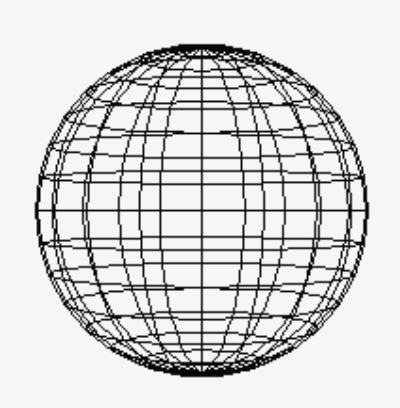
```
glDisable(GL_CLIP_PLANEi);
```

ملاحظة :

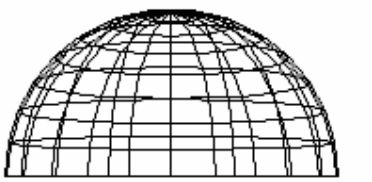
تعيد الـــ OpenGL بناء حواف النموذج الذي تم اقتطاعه بشكل آلي .

مثال:

بفرض لدينا الشكل التالي :



ونريد إجراء قطع لنصف السفلي

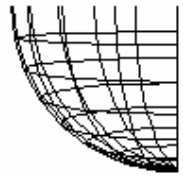


، وقطع لنصف اليساري فنحصل على الشكل التالي :

```
void display(void)
{
 GLdouble eqn0[4] = {0.0, 1.0, 0.0, 0.0};
 GLdouble eqn1[4] = {1.0, 0.0, 0.0, 0.0};

 glClear(GL_COLOR_BUFFER_BIT);
 glColor3f (1.0, 1.0, 1.0);

 glPushMatrix();
 gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    */   clip lower half -- y < 0/*
 glClipPlane (GL_CLIP_PLANE0, eqn0);
 glEnable (GL_CLIP_PLANE0);

    */   clip left half -- x < 0/*
 glClipPlane (GL_CLIP_PLANE1, eqn1);
 glEnable (GL_CLIP_PLANE1);

 glRotatef (90.0, 1.0, 0.0, 0.0);
 glutWireSphere(1.0, 20, 16);
 glPopMatrix();
 glFlush ();
}// end display
```

تمرين :

أعد كتابة المثال السابق بحيث يظهر الشكل التالي :



اجري عملية القطع على المحور z و لاحظ الخرج .

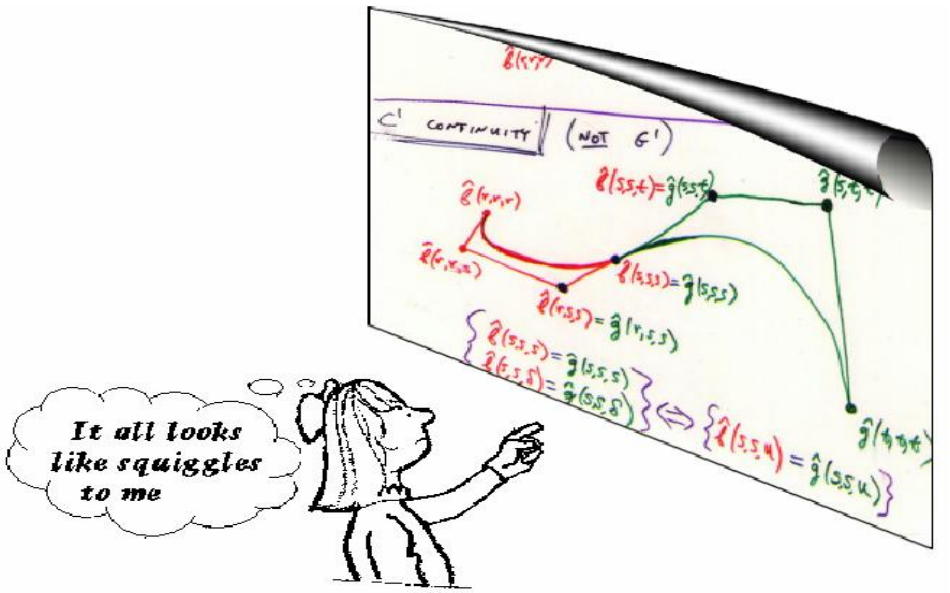
الفصل الخامس

السطوح والمنحنيات

مقدمة:

تتألف المنحنيات والسطوح من أغراض ذات خطوط مستقيمة، وقد يكون توصيفها بمعادلة رياضية ممكناً أو غير ممكن ولكن يمكن تصميم أي منحني عن طريق استعمال نقاط التحكم. كلما زادت نقاط التحكم كلما زادت نعومة المنحني ودقته .

منحني بيزييه Bezier curve: خط مولد رياضياً يمثل منحنيات غير نظامية. وعملية التوليد تتم وفق خوارزميات معينة. وقد كان العالم الفرنسي الرياضي بيير بيزييه أول من قدم وصفاً لهذه المنحنيات وفق خوارزميات. وتتمتع منحنيات بيزييه بأنها لا تحتاج إلى أكثر من بضع نقاط لكي تعرف عدداً كبيراً من الأشكال غير النظامية. وفي البرامج الرسومية تستخدم منحنيات بيزييه لرسم أشكال حرة غير نظامية ناعمة (ملساء) وسطوح واصلة بين نقطتين.



نستخدم تعليمات الـــ **OpenGL's evaluator** لرسم المنحنيات والسطوح.

الـــ Evaluators: توفر طريقة لتحديد النقاط على المنحني أو السطح أو على جزء منهما باستخدام نقاط التحكم. ومن ثم يمكن رسم المنحني أو السطح بشكل واضح ودقيق. يمكن توظيف النقاط التي تولدها عملية الـــ Evaluator لعدة استخدامات منها :

١- تحديد المكان المطلوب لرسم السطح،
٢- رسم نموذج خطي للسطح،
٣- وكذلك لرسم سطوح مضيئة ومظلله.

تعطى معادلة المنحني بشكل عام كالتالي:

$$\mathbf{C}(u) = [\mathbf{X}(u)\ \mathbf{Y}(u)\ \mathbf{Z}(u)]$$

حيث : يأخذ المتحول u قيم مختلفة من مجال ما.

تعطى معادلة السطح بشكل عام كالتالي:

$$\mathbf{S}(u,v) = [\mathbf{X}(u,v)\ \mathbf{Y}(u,v)\ \mathbf{Z}(u,v)]$$

حيث تأخذ كل من u, v قيم مختلفة من مجالات مختلفة.

من أجل كل نقطة u تعطي الصيغة c(u) أو s(u) نقطة من نقاط المنحني أو السطح. ولاستخدام الـــ evaluator يجب في البداية تعريف الصيغتين c(u) أو s(u) ومن ثم استخدام التعليمة **glEvalCoord1** أو **glEvalCoord2** بدلاً من **glVertex**. ويمكن التعامل مع عقد المنحني أو السطح كغيرها من العقد vertices حيث يمكن استخدامها لتشكيل نقاط وخطوط . وهناك تعليمات أخرى مسؤولة عن توليد سلسلة من هذه العقد والتي ينتج عنها في النهاية شبكة منتظمة ضمن مجال u.

١. One-Dimensional Evaluators :

تعطى معادلة منحني بزييه بالعلاقة التالية:

$$C(u) = \sum_{i=0}^{n} B_i^n(u) P_i$$

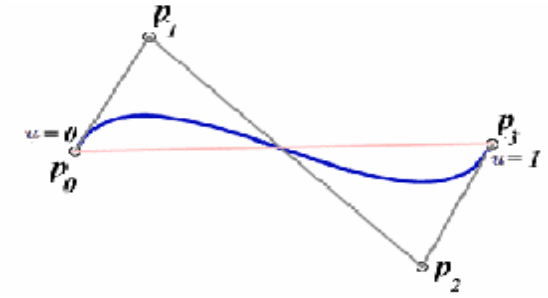
يعطى تابع الانحناء Bernstein من الدرجة n بالعلاقة التالية:

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}$$

Pi تمثل قيم نقطة التحكم .

مثال ١ :

نستخدم فيه one-dimensional evaluators لرسم منحني بيزييه مستخدمين فيه أربع نقاط تحكم.

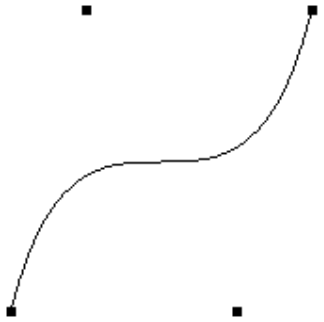


Example 11-1 : Drawing a Bézier Curve Using Four Control Points: bezcurve.c

```
#include <GL/gl.h>
#include <GL/glu.h>
#include "aux.h"

GLfloat ctrlpoints[4][3] = {
    { -4.0, -4.0, 0.0}, { -2.0, 4.0, 0.0},
    {2.0, -4.0, 0.0}, {4.0, 4.0, 0.0}};

void myinit(void)
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4,&ctrlpoints[0][0]);
    glEnable(GL_MAP1_VERTEX_3);
    glShadeModel(GL_FLAT);
}

void display(void)
{
    int i;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINE_STRIP);
        for (i = 0; i <= 30; i++)
            glEvalCoord1f((GLfloat) i/30.0);
    glEnd();
   /* The following code displays the control points as dots. */
    glPointSize(5.0);
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_POINTS);
        for (i = 0; i < 4; i++)
            glVertex3fv(&ctrlpoints[i][0]);
    glEnd();
    glFlush();
}
```

```
void myReshape(GLsizei w, GLsizei h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        glOrtho(-5.0, 5.0, -5.0*(GLfloat)h/(GLfloat)w,
            5.0*(GLfloat)h/(GLfloat)w, -5.0, 5.0);
    else
        glOrtho(-5.0*(GLfloat)w/(GLfloat)h,
            5.0*(GLfloat)w/(GLfloat)h, -5.0, 5.0, -5.0, 5.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
void main(int argc, char** argv)
{
glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize (600, 450);
   glutCreateWindow(argv[0]);
   myinit();
   glutReshapeFunc(reshape);
   glutDisplayFunc(display);
   glutMainLoop();
}
```

يكون الخرج :



في هذا المثال رسمنا منحني بزييه بالاستعانة بأربعة نقاط تحكم تم تعيينها في المصفوفة ctrlpoints[][] واستخدمت هذه المصفوفة كبارمتر للتعليمة glMap1f حيث تأخذ هذه التعليمة البارمترات التالية:

- **GL_MAP1_VERTEX_3** : لتوليد عقد ثلاثية البعد .
- **البارمتر ذو القيمة 0:** الحد الأدنى للمجال الذي تنتمي إليه قيم المتحول u.
- **البارمتر ذو القيمة ١:** الحد الأعظم للمجال الذي تنتمي إليه قيم المتحول u.
- **البارمتر ذو القيمة ٣:** عدد النقاط المولدة بين كل نقطة تحكم و نقطة التحكم التي تليها ، حيث تأخذ النقاط المولدة قيماً ذات فواصل عشرية.
- **البارمتر ذو القيمة ٤:** يحدد درجة المنحي وتعطى درجة المنحني بالعلاقة degree+1 =n حيث n هنا تساوي 4 فبالتالي درجة المنحي تساوي ٣ .
- **&ctrlpoints[0][0]** : مصفوفة نقاط التحكم.

وتفيد التعليمة glEnable في تفعيل one-dimensional evaluator ، ويتم رسم المنحي ضمن الإجرائية display() ضمن التعليمتتي glBegin()،glEnd() . وبما أننا قمنا بتفعيل عملية الـ evaluator فإن التعليمة **glEvalCoord1f()** ستعمل بشكل مشابه تماماً لعمل التعليمة glVertex() ويكون بارمتر هذه التعليمة عبارة عن إحدى قيم المتحول u الذي ينتمي للمجال [0,1].

٢. Two-Dimensional Evaluators :

تتشابه المفاهيم هنا مع تلك التي تعرفنا عليها سابقاً في الـ One-Dimensional Evaluators باستثناء أنه هنا سيتم التعامل مع بارمترين هما u,v بدلاً من بارمتر واحد.

تعطى الصيغة الممثلة لسطوح بيزييه بالعلاقة التالية :

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(u)\, B_j^m(v)\, P_{ij}$$

حيث Pij : مجموعة قيم نقاط التحكم .

Bi : نفس التابع الذي مر معنا سابقاً.

نعرف الــ evaluator بالتعليمة glMap2*() . ونفعل هذه العملية باستخدام التابع glEnable ثم نحقق عملية رسم السطح باستخدام التعليمة glEvalCoord2() التي يتم استدعاؤها بين التعليمتين glBegin()،glEnd().

void **glEvalCoord2**{fd}{v}(**TYPE u, TYPE v**);

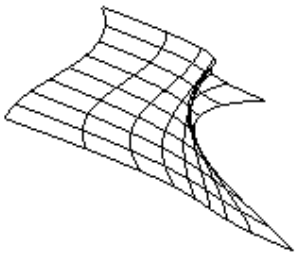
مثال ٢:

نستخدم فيه tow-dimensional evaluators لرسم سطح بيزييه.

Example: Drawing a Bézier Surface: bezsurf.c

```
#include <GL/gl.h>
#include <GL/glu.h>
#include "aux.h"

GLfloat ctrlpoints[4][4][3] = {
    {{-1.5, -1.5, 4.0}, {-0.5, -1.5, 2.0},
        {0.5, -1.5, -1.0}, {1.5, -1.5, 2.0}},
    {{-1.5, -0.5, 1.0}, {-0.5, -0.5, 3.0},
        {0.5, -0.5, 0.0}, {1.5, -0.5, -1.0}},
    {{-1.5, 0.5, 4.0}, {-0.5, 0.5, 0.0},
        {0.5, 0.5, 3.0}, {1.5, 0.5, 4.0}},
    {{-1.5, 1.5, -2.0}, {-0.5, 1.5, -2.0},
        {0.5, 1.5, 0.0}, {1.5, 1.5, -1.0}}
};

void display(void)
{
    int i, j;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glPushMatrix ();
    glRotatef(85.0, 1.0, 1.0, 1.0);
    for (j = 0; j <= 8; j++) {
        glBegin(GL_LINE_STRIP);
```

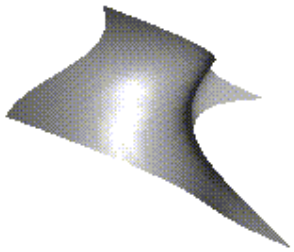
ويكون الخرج كالتالي:



مثال٣:

وسنعرض في هذا المثال سطح بيزييه بشكل مضاء ، مستخدمين تعليمتين جديدتين لم نتحث عنهمـــا ســـابقاً هما :

glMapGrid2,glEvalMesh2 (تعرف بمفردك على هاتين التعليمتين) .

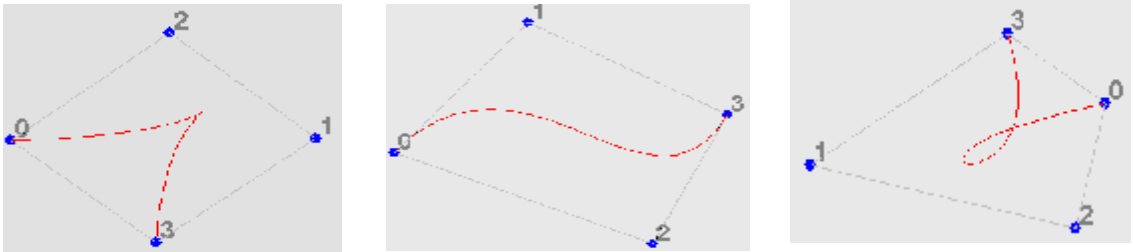


Example: Drawing a Lit, Shaded Bézier Surface Using a Mesh: bezmesh.c

```
void initlights(void)
{
    GLfloat ambient[] = { 0.2, 0.2, 0.2, 1.0 };
    GLfloat position[] = { 0.0, 0.0, 2.0, 1.0 };
    GLfloat mat_diffuse[] = { 0.6, 0.6, 0.6, 1.0 };
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_shininess[] = { 50.0 };

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
    glLightfv(GL_LIGHT0, GL_POSITION, position);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT_AND_BACK,GL_SHININESS, mat_shininess);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
        glRotatef(85.0, 1.0, 1.0, 1.0);
        glEvalMesh2(GL_FILL, 0, 8, 0, 8);
```

تمرين :

اكتب تطبيق لرسم منحني بيزييه بحيث يتم تحديد نقاط التحكم من قبل المستخدم بواسطة الماوس ثــم يرســم المنحني ولاحظ الأشكال المختلفة لمنحني بيزييه بحسب اختلاف نقاط التحكم وإليك بعض الأمثلة :



الفصل السادس

الشفافية والضبابية

مقدمة:

قدمت لنا الـــ OpenGL الكثير من الحلول لرسم الصور وتجسيد المناظر الطبيعية، فكانت توابع الحركـــة التي ساعدت على التحكم بمكان ظهور الأشكال وطريقة توضعها، وكانت توابع الرؤية التي حددت الزاويـــة التي يُنظر من خلالها للمشهد. كما استطعنا التمييز بين نوعيين من الإسقاط (المتوازي والمتلاشي) .

وعندما تراكبت السطوح مع بعضها وفرت الـــ OpenGL توابع لاقتطاع وإخفاء السطوح. وهنـــا ســـنقف لنتساءل لماذا اقتطعنا الجزء المتراكب من السطح ؟ لماذا كانت النتيجة الطبيعية للتراكب هي اقتطاع وإخفاء ذلك الجزء ؟! الجواب : لأن كل السطوح والأشكال التي تعاملنا معها حتى هذه اللحظة كانـــت ذات طبيعـــة مصمتة وغير نفوذه . ولكن في العالم الحقيقي الأمر مختلف، حيث تتنوع طبيعة المـــواد والـــسطوح فمنهـــا الشفاف ونصف الشفاف والمصمت. فإذا تراكب سطحان شفافان و ملونان لن يخفي أحدهما الآخـــر وإنمـــا سيظهر الجزء المتراكب بلون مركب من لون السطحين. وبما أن الـــ OpenGL تهـــتم بتجـــسيد النمـــاذج الواقعية إلى حد كبير كان لابد لها من تقديم توابع تخدم هذه الفكرة " **الشفافية والمـــزج اللـــوني للـــسطوح الشفافة المتراكبة**".

أما من ناحية أخرى فقد وفرت لنا الـــ OpenGL توابع لإعطاء المشاهد الطبيعية ألوان حقيقيـــة وإضـــاءة رائعة. فاستطاعت إظهارها بألوان زاهية وبإضاءة براقة أو بألوان قاتمة وإضاءة كامدة أو مظلمة. ولكن ماذا عن الأحوال الجوية ؟ ماذا لو كنا نريد تجسيد وتمثيل منظر طبيعي في يوم ضبابي؟ كيـــف ســـتعالج الـــــ OpenGL موضوع "**الضباب**" ؟!.

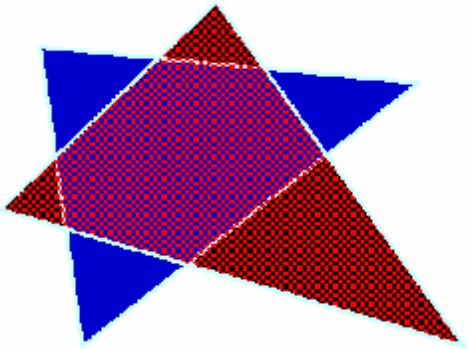
سنناقش فيما يلي هاتين الفكرتين (الشفافية والمزج اللوني، الضبابية)، وسنرى ما هي التوابع التي تقـــدمها الـــ OpenGL لتحقيق ذلك.

▫ الشفافية والمزج اللوني :

الشفافية:

لنتذكر معاً تابع التلوين glColor*() الذي يأخذ أربعة وسطاء RGBA حيث كانت الـR تمثل مركّبة اللون الأحمر، والـ G تمثل مركّبة اللون الأخضر، والـB تمثل مركّبة اللون الأزرق . أما بالنسبة للمركبة الرابعة A فكنا ندعوها ألفا وكنا نعطيها دائماً قيمة افتراضية = ١,٠ و لكن لم نشرح لماذا هذه القيمة بالذات ولماذا لم نكن نغيرها في كل الأمثلة التي مرت معنا سابقاً . كما أننا رأينا هذه المركبة عندما درسنا المصادر الضوئية وعندما حددنا خصائص المواد . ولكن ما الفائدة الحقيقة وراء القيمة ألفا ؟

تحدد قيمة المركبة ألفا درجة شفافية السطح المرسوم، فعندما تكون قيمتها تساوي الواحد هذا يعني أن السطح مصمت وغير نفوذ وكلما قلت قيمة المركبة ألفا كلما زادت شفافية سطح الشكل المرسوم. كما أنها تلعب دور هام في عملية المزج اللوني. فبفرض لدينا سطحين شفافين الأول لونه أحمر والثاني لونه أزرق فإذا حصل تراكب بين هذين السطحيين سيظهر الجزء المتراكب بلون بنفسجي مع العلم أن شدة لون الجزء المتراكب تختلف بين فاتح وغامق بحسب شفافية السطح الأمامي فإذا كانت شفافية السطح الأمامي كبيرة فإن جزءاً كبيراً من لون السطح الخلفي سيظهر للعيان .



المزج اللونى :

تتم عملية المزج اللوني بعد الانتهاء من مسح الصورة وتحويلها إلى مجموعة بكسلات ولكن قبل تخزينها في المخزن المؤقت المسمى " Framebuffer ". مع العلم أن كل بكسل من هذه البكسلات تحمل مجموعة من البيانات تتعلق بـ لون البكسل، إحداثياته، ...الخ . وتسمى هذه البكسلات الحاملة للبيانات بـ Fragments . حيث تتم عملية مسح نقاط الصورة الأولى ووضعها في المخزن المؤقت ومن ثم مسح نقاط الصورة الثانية، و قبل تخزين بكسلات الصورة الثانية نميز بين حالتين :

١. الحالة الأولى: إذا لم يكن لدينا مركبة مثل ألفا عندها سيتم وضع نقاط الصورة الثانية المشتركة بالإحداثيات مع نقاط الصورة الأولى فوقها مسبباً إخفاء جزء من الصورة الأولى .
٢. الحالة الثانية: في حالة وجود المركبة ألفا فإن وضع بكسلات من الصورة الثانية فوق بكسلات الصورة الأولى في المخزن المؤقت لن يسبب إخفاء أو ضياع لأي بكسل من أحدهما وبالتالي عند عرض الصورة على الشاشة ستظهر جميع البكسلات من كلتا الصورتين وبحسب الشفافية التي حددت بالقيمة ألفا ستظهر بكسلات منطقة التراكب بلون مركب من لوني البكسلات المتوضعة فوق بعضها .

المصدر والوجهة:

يستخدم مصطلحي المصدر والوجهة للدلالة على الجسميين المتراكبين، حيث يسمي الجسم الذي تم رسمه أولاً ووضعت بكسلاته في المخزن المؤقت **بالوجهة Destination**، أما الجسم الذي سيتم رسمه لاحقاً والـــذي ستوضع بعض بكسلاته فوق بكسلات الأول فيسمى **بالمصدر Source**.

ملاحظة هامة :

لا يمكن الاستفادة من القيمة ألفا عند استخدام نظام الألوان color-index وإنما يمكن رؤية تأثير القيمة ألفا في نظام الألوان RGBA فقط.

لون منطقة تراكب:

إن المفهوم العام للمزج اللوني يقول: بأن لون السطح الناتج عن تراكب شكلين هو تركيب من لون السطحيين ويعطى بالشكل التالي :

$$O(rgb) = S(rgb) * D(rgb)$$

حيث `S(rgb)` هو لون المصدر، `D(rgb)` هو لون الوجهة، و O(rgb) هو لون الخرج.
بحيث تطبق هذه المعادلة كالتالي :

$$O(r) = S(r) * D(r)$$

$$O(g) = S(g) * D(g)$$

$$O(b) = S(b) * D(b)$$

ولكن هذا صحيح فقط عندما تشابه الخصائص المادية لكلا السطحين، في حين عند اختلاف هذه الخصائص بحيث يكون أحد الجسمين شفاف أكثر من الآخر عندها لن يكون اللون الناتج مجرد مزيج من اللونيين وإنما سيغلب على اللون الناتج لون أحد السطحيين بحسب نفوذيتهما ولكن هذا لا يتحقق بمجرد ضرب القيميتين ألفا لكلا الجسمين ببعضهما أي لا يمكن الحصول على درجة نفوذية الجزء المتراكب من خلال ضرب القيميتين ألفا فقط. إذاً هناك معادلة رابعة يجب أخذها بعين الاعتبار وهي معادلة تحديد معامل المزج اللوني بحيث نحدد فيها أي من السطحين سيغلب لونه على منطقة التراكب . لنرمز لهذا المعامل بالرمز P، يأخذ هذا المعامل مجموعة قيم ثابتة هي :

- Zero
- One
- Source Color
- Source Alpha
- Inverse Source Color
- Inverse Source Alpha
- Destination Color

- Destination Alpha
- Inverse Destination Color
- Inverse Destination Alpha

حيث يقابل كل ثابت من الثوابت السابقة بالمركبات الثلاثة rgb، فمثلاً :

P(Zero) = P(rgb) = P(000)

P(One) = P(rgb) = P(111)

وهكذا ...الخ .

وتعطى معادلة اللون النهائي بعدة طرق بحسب نوع المزج المستخدم :

- مزج لوني ناتج عن عملية الجمع تكون معادلة المزج اللوني كالتالي :

O(rgba) = (S(rgba) * P0(rgba)) + (D(rgba) * P1(rgba))

- مزج لوني ناتج عن عملية الطرح تكون معادلة المزج اللوني كالتالي :

O(rgba) = (S(rgba) * P0(rgba)) - (D(rgba) * P1(rgba))

- مزج لوني ناتج عن عملية طرح معاكسة تكون معادلة المزج اللوني كالتالي :

O(rgba) = (D(rgba) * P1(rgba)) - (S(rgba) * P0(rgba))

حيث يترك للمستخدم اختيار قيمة كل من P1، P0 من مجموعة الثوابت التي عرضناها سابقاً مع مراعاة الشرط التالي :
لا يمكن أن يأخذ المعامل P0 نفس قيمة لون المصدر، وكذلك لا يسمح بأن يأخذ المعامل P1 نفس لون الوجهة.

ملاحظة:

قد يتساءل البعض عن سبب تنوع عمليات المزج اللوني، إن هذا التنوع يتيح لنا إظهار المزج اللوني المماثل للواقع والذي نراه في حياتنا اليومية عند تراكب سطحين نفوذ ين، بالإضافة إلى إظهار أنواع أخرى من المزج التي قد تستخدم لإجراء عمليات خداع للبصر أو إظهار بعض اللوحات الفنية الجميلة بحيث نستخدم مزجاً يعطي تراكباً لونياً جميلاً وغير متوقع.

المزج اللوني في الـــ OpenGL:

تستخدم الـــ OpenGL طريقة الجمع لإجراء عملية المزج اللوني

O(rgba) = (S(rgba) * P0(rgba)) + (D(rgba) * P1(rgba))

حيث تستخدم التابع التالي :

glBlendFunc(GLenum *sfactor*, GLenum *dfactor*)

يمثل الوسيط sfactor قيمة المعامل P0 ، يمثل الوسيط dfactor قيمة المعامل P1 .
وكما سبق وشاهدنا أن المعامل P يأخذ مجموعة قيم ثابتة فإن الـــ OpenGL تقدم مجموعة الثوابت التالية لهذا المعامل. سنعرضها بالجدول التالي :

الثوابت	القيم التي تقابلها
GL_ZERO	(0, 0, 0, 0)
GL_ONE	(1, 1, 1, 1)
GL_DST_COLOR	(Rd, Gd, Bd, Ad)
GL_SRC_COLOR	(Rs, Gs, Bs, As)
GL_ONE_MINUS_DST_COLOR	(1, 1, 1, 1)-(Rd, Gd, Bd, Ad)
GL_ONE_MINUS_SRC_COLOR	(1, 1, 1, 1)-(Rs, Gs, Bs, As)
GL_SRC_ALPHA	(As, As, As, As)
GL_ONE_MINUS_SRC_ALPH A	(1, 1, 1, 1)-(As, As, As, As)
GL_DST_ALPHA	(Ad, Ad, Ad, Ad)
GL_ONE_MINUS_DST_ALPH A	(1, 1, 1, 1)-(Ad, Ad, Ad, Ad)
GL_SRC_ALPHA_SATURATE	(f, f, f, 1); f=min(As, 1-Ad)

يكون عادة المعامل P1 متتم للمعامل P0 .
فعندما يكون P0 = GL_SRC_ALPHA فإن P1 = GL_ONE_MINUS_SRC_ALPH A
وبالتالي أن قيمة ألفا المختارة في تابع التلوين للكائن المصدر تلعب دورا هاما هنا، ولكن إذا كان P0 = One، P1 = Zero عندها مهما كانت قيمة ألفا المختارة في تابع التلوين لن يكون لها أي أثر وسيغلب بجميع الأحوال لون المصدر على الوجهة و سيسبب إخفاء الجزء المتراكب من الوجهة. ولا يوجد ما يمنع المبرمج

من اختيار قيم لـــ P1،P0 دون أن يكون بينهما أي علاقة بمعنى دون أن يكون P1 متتم لـــ P0 سوى أنه قد لا يحصل على النتيجة المرغوبة في حال اتباعه أسلوب عشوائي في اختيار قيم P1، P0 .
ويكون استدعاء التابع glBlendFunc بالشكل التالي :

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glBlendFunc(GL_ONE, GL_ZERO);
```

يجـــب التنويـــه إلـــى أن عمليـــة المـــزج اللـــوني لا تـــتم مـــا لـــم نقـــوم بتفعيلهـــا باســـتخدام التـــابع **glEnable(GL_BLEND) ;** .

والآن لنأخذ مثال نرى فيه كيف يمكن إجراء المزج اللوني في الـــ OpenGL .

مثال:

بفرض أنه لدينا سطحين متراكبين أحدهما لونه أزرق والآخر لونه أصفر، وبفرض أن السطح الأصفر هو الوجهة وأن السطح الأزرق هو المصدر وبفرض أن قيمة ألفا للمصدر تساوي ٠,٧٥ ، وبفرض أن تابع المزج اللوني يأخذ القيم التالية :

```
glBlendFunc (GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA );
```

عندها سيكون اللون الغالب على منطقة التراكب هو اللون الأزرق . وسيكون الخرج كالتالي :

وسنعرض فيما يلي النص البرمجي :

```
# #include "stdafx.h"

#include <GL\glut.h>    // Header File For GLUT Library
#include <GL\GLAUX.h>  // Header File For The Glaux Libra
#include <GL\GLU.h>
#include <GL\GL.h>

  */Initialize alpha blending function.
 /*
static void init(void)
{
   glEnable (GL_BLEND);
   glBlendFunc (GL_SRC_ALPHA,  GL_ONE_MINUS_SRC_ALPHA );
   glShadeModel (GL_FLAT);
   glClearColor (0.0, 0.0, 0.0, 0.0);
}

static void drawLe
{
 */   draw yellow

   glBegin (GL_TRI
      glColor4f(1.
      glVertex3f(0
      glVertex3f(0
      glVertex3f(0
   glEnd();
}
```

```
static void drawRightTriangle(void)
{
 */   draw cyan triangle on RHS of screen/*

   glBegin (GL_TRIANGLES);
      glColor4f(0.0, 1.0, 1.0, 0.75);
      glVertex3f(0.9, 0.9, 0.0 );
      glVertex3f(0.3, 0.5, 0.0);
      glVertex3f(0.9, 0.1, 0.0);
   glEnd();
}
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);

   drawLeftTriangle();
   drawRightTriangle();

   glFlush();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
      gluOrtho2D (0.0, 1.0, 0.0, 1.0*(GLfloat)h/(GLfloat)w);
   else
      gluOrtho2D (0.0, 1.0*(GLfloat)w/(GLfloat)h, 0.0, 1.0);
}

  */Main Loop
  * Open window with initial window size, title bar ،
  * RGBA display mode, and handle input events.
/*

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (200, 200);
   glutCreateWindow (argv[0]);
   init();
   glutReshapeFunc (reshape);
   glutDisplayFunc (display);
   glutMainLoop();
   return 0;
```

وبفرض لو أننا قمنا برسم الكائن الأزرق بداية بحيث يكون هو الوجهة، ثم رسمنا

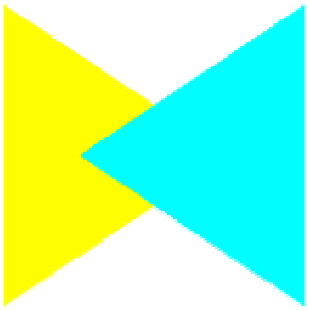
السطح الأصفر بحيث يكون هو المصدر . وإذا لم نقم بتغيير تابع المزج عندها سيكون اللون الغالب على منطقة المزج هو اللون الأصفر وسيكون الخرج كالتالي :

وهذا يعني أن ترتيب رسم الأجسام سيسبب اختلاف في لون منطقة التراكب .

ولو أننا استخدمنا تابع المزج اللوني بالشكل التالي :

```
glBlendFunc (GL_ONE, GL_ZERO );
```

ولو أننا عدنا للحالة الأولى بحيث يكون السطح الأزرق هو المصدر فعندها لن يكون هناك أي أهمية لقيمة ألفا المختارة في تابع التلوين لا للمصدر ولا للوجهة وسيكون الخرج كالتالي :



تمرين :

أعد كتابة المثال السابق بحيث نغير المصدر والوجهة عن طريق التحكم بلوحة المفاتيح، بحيث يكون السطح الأزرق هو الوجهه، والأصفر هو المصدر ثم باستخدام أحد أزرار لوحة المفاتيح يصبح السطح الأزرق هو المصدر والأصفر هو الوجهة ولاحظ الفرق في الخرج الناتج .

▫ الضبابية:

تعتبر الأحوال الجوية أحد أهم المواضيع التي يجب الاهتمام بها عند تجسيد المناظر الطبيعية، وقد استطعنا تمثيلها سابقاً باستخدام الألوان والإضاءة فصورنا الطبيعة في الأيام المشمسة حيث استخدمنا الألوان الزاهية والإضاءة البراقة، وكذلك في الأيام المعتمة باستخدام ألوان قاتمة وإضاءة كامدة. ولكن هناك حالة جوية أخرى لم نأتي على ذكرها وهي الضباب . حيث يؤثر الضباب على المشاهد الطبيعية فيعطي غشاوة وعدم وضوح للمشهد . كما يمكن استخدام الضباب في حالات أخرى غير الأحوال الجوية فمثلاً يمكن استخدامه في النماذج التي تظهر فيها الكائنات تدريجياً لتصبح واضحة وبالعكس في النماذج التي تختفي وتتلاشى فيها الكائنات تدريجياً . وكذلك يعرّف الدخان على أنه نوع من الضباب، عندها نستفيد من الضباب لتصوير حريق ما أو عندما نريد تصوير مشهد للتلوث الناتج عن دخان السيارات والمصانع.

لاحظ من الشكل كيف أن الضباب يجعل الأجسام تبدو وكأنها تبتعد عن العين وتندمج مع الخلفية.

تقدم الـ OpenGL تابع يمكنها من تمثيل الضباب، هذا التابع هو :

```
void glFog{if}{v}(GLenum pname, TYPE param);
```

ولرؤية أثر هذا التابع يجب تفعيله في البداية باستخدام التعليمة التالية:

```
glEnable(GL_FOG);
```

كما تتيح الـ OpenGL القدرة على التحكم بآلية توليد الضباب باستخدام التابع :

```
void glHint(GLenum target, GLenum hint);
```

فإما أن يتم توليده من أجل كل pixel أو من أجل كل vertex، حيث يتم اختيار ذلك بحسب ما يتطلبه التطبيق من دقة في الإظهار أو سرعة في التنفيذ. فعندما نكون بحاجة لدقة في الإظهار يكون الوسيط hint مساوياً لـ GL_NICEST وعندما نكون بحاجة لسرعة في التنفيذ يكو ن الوسيط hint مساوياً لـ GL_FASTEST . ونعطي هذا الوسيط القيمة GL_DONT_CARE فيما عدا ذلك أي عندما لا نكون مهتمين بأي من الحالتين السابقتيين.

أما فيما يتعلق بالوسيط target فإنه يأخذ القيمة GL_FOG_HINT في الحالات الثلاث السابقة .

```
glHint (GL_FOG_HINT, GL_DONT_CARE);
```

```
glHint (GL_FOG_HINT, GL_FASTEST );
glHint (GL_FOG_HINT, GL_NICEST );
```

كما يمكن اختيار لون ما للضباب، حيث سبق وقلنا أن الضباب ليس مجرد تعبير عن حالة جوية وإنما يمكن استخدامه في تطبيقات أخرى . مع العلم أن هذا اللون يمكن تحديده سواءً عند استخدام نظام الألـوان الـــ RGBA أو النظام color-index مع وجود بعض الفروقات .

و يجب الانتباه إلى أن استخدام تابع الضباب في التطبيقات الرسومية يتم قبل رسم النموذج وإجراء أي عملية تحويل عليه حيث تحدد خصائصه في إجرائية الـ init() مثله مثل الإضاءة التي غالبـاً مـا تـضبط خصائصها في إجرائية الـ init() .

المعادلات الرياضية المتعلقة بحساب كثافة الضباب:

يمكن التحكم بكثافة الضباب في المشهد استناداً إلى ثلاثة معادلات رياضية :

$$f = e^{-(density \cdot z)} \qquad \text{(GL\_EXP)}$$

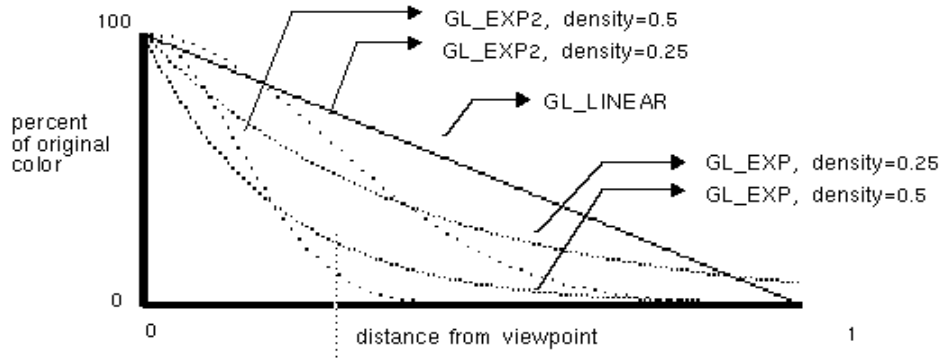
$$f = e^{-(density \cdot z)^2} \qquad \text{(GL\_EXP2)}$$

$$f = \frac{end - z}{end - start} \qquad \text{(GL\_LINEAR)}$$

حيث:

- Start: بداية المنطقة الخاضعة للضباب.
- End: أبعد نقطة تقع تحت تأثير الضباب ويمكن رؤيتها.
- Density: درجة أو معامل كثافة الضباب.
- z : البعد بين منتصف النقطة المعتبرة من الشكل وعين المراقب .

و تنحصر قيم التابع f ضمن المجال [0,1].



يوضح هذا الشكل الاختلاف في كثافة الضباب في كل من المعادلات الثلاثة السابقة .

يتم تحديد نوع المعادلة الرياضية المستخدمة عند تحديد وسطاء التابع

```
void glFog{if}{v}(GLenum pname, TYPE param);
```

فمثلاً :

عندما نريد استخدام المعادلة الرياضية الأولى يأخذ الوسيط pname القيمة GL_FOG_MODE ، ويأخذ الوسيط param القيمة GL_EXP . فيكون شكل التابع glFog*() كالتالي:

```
glFogi(GL_FOG_MODE,GL_EXP);
```

ولتحديد قيمة المتحول density المستخدم في المعادلة نستخدم أيضاً التابع glFog*() حيث يأخذ الوسيط pname القيمة GL_FOG_DENSITY و بالوسيط param نعطي الكثافة قيمة ما . كما في الشكل التالي:

```
glFogf(GL_FOG_DENSITY,1.00);
```

وبالمثل يتم استخدام باقي المعادلات حيث سنوضح القيم التي يمكن أن يأخذها هذين الوسيطين بالجدول التالي:

Pname	**Param**	**Meaning**
GL_FOG_MODE	GL_EXP, GL_EXP2, GL_LINEAR	the fog function
GL_FOG_DENSITY	density	density value in the fog function.
GL_FOG_STARET	start	start distance in the fog function.
GL_FOG_END	end	end distance in the fog function.
GL_FOG_INDEX	index	the fog color (color-index)

تعتبر عملية مزج الألوان القاعدة الأساسية التي يتم تطبيقها لإظهار أثر لون الضباب في المشهد حيث يتم إجراء مزج بين لون الضباب ولون الأجسام المرسومة في النموذج .وذكرنا أنه يمكن استخدام الضباب في

نظامي الألوان الـــ RGBA، الـــ color-index ، مع بعض الاختلاف المتعلقة بطريقة إجراء المزج ففي نظام الألوان الـــ RGBA يحدد اللون النهائي الناتج عن عملية المزج وفق المعادلة التالية:

$$C_{Final} = f * C_{Current} + (1 - f) * C_{Fog}$$

حيث $C_{Current}$: عبارة عن لون بكسلات المصدر (ناشئة عن الأجسام في النموذج).
C_{Fog} : عبارة عن لون بكسلات الوجهة (ناشئة عن الضباب في النموذج).

أما f فهو عبارة عن إحدى المعادلات الرياضية الثلاثة التي تحدد كثافة الضباب، وبالتالي فإن لون الضباب يختلف باختلاف كثافته .

أما في نظام الألوان color-index فيحسب وفق المعادلة التالية:

$$I = Ii + (1 - f) If$$

حيث Ii : عبارة عن رقم اللون في جدول الألوان والمرتبط بالمصدر (ناشئة عن الأجسام في النموذج).
If : عبارة عن رقم اللون في جدول الألوان و المرتبط بالوجهة (ناشئة عن الضباب في النموذج).

ولتحديد طريقة الحساب التي يجب اعتمادها في الـــ OpenGL نستخدم التابع

```
glFogfv(GL_FOG_COLOR,color);
```

حيث يكون الوسيط color عبارة عن شعاع مؤلف من أربع مركبات في نظام الألوان RGBA كل منهـــا قيمة مركبة من الـــ RGBA، ويعرف كالتالي:

```
GLfloat color[4]= {0.70,0.70,0.70,1.0};
```

أما في نظام الألوان color-index نستخدم التابع

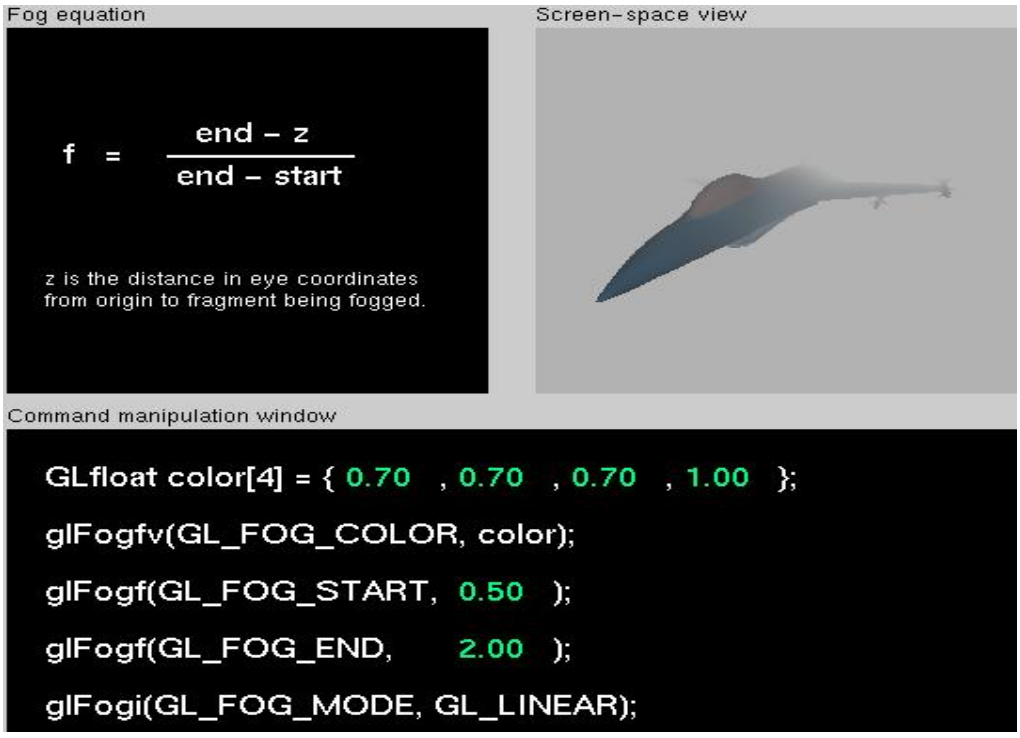
```
glFogfv(GL_FOG_INDEX,color);
```

ولكن في هذه الحالة يكون الوسيط color عبارة عن قيمة وحيدة تشير إلى رقم اللون في جدول الألوان .

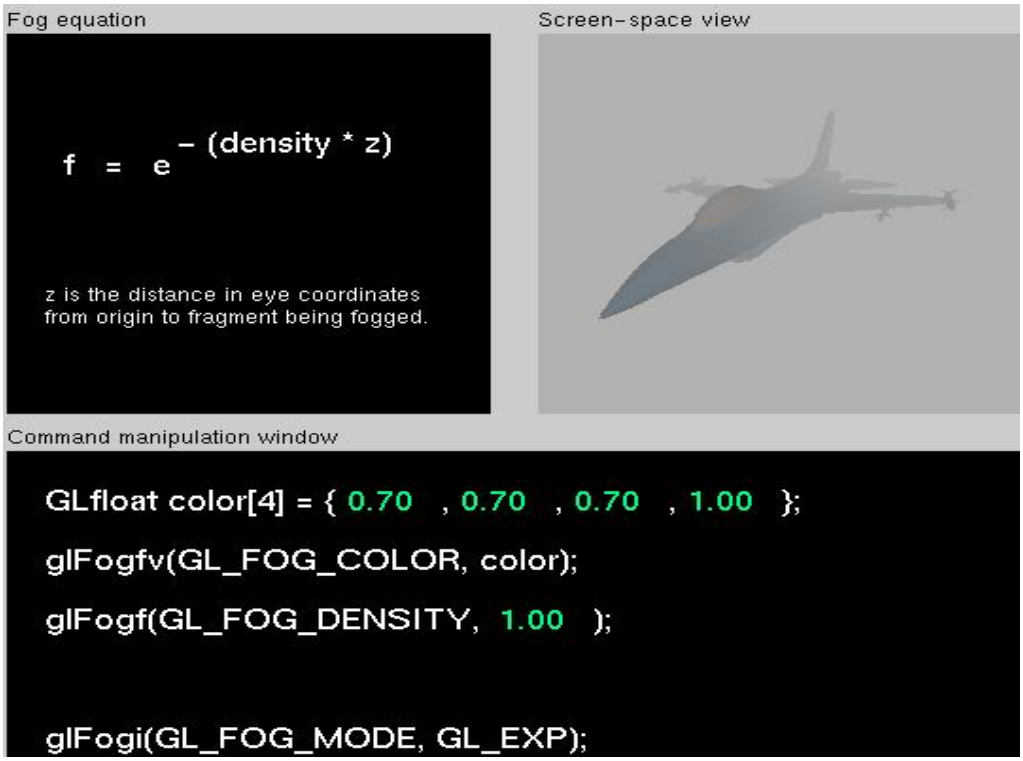
أمثلة :

سنعرض بعض الأمثلة التي تبين الفرق بين كل شكل من أشكال المعادلات ، مع الإشارة إلى أن نظام الألوان المستخدم في جميع الأمثلة هو النظام RGBA .
بفرض لدينا مشهد يعرض طائرة في السماء .

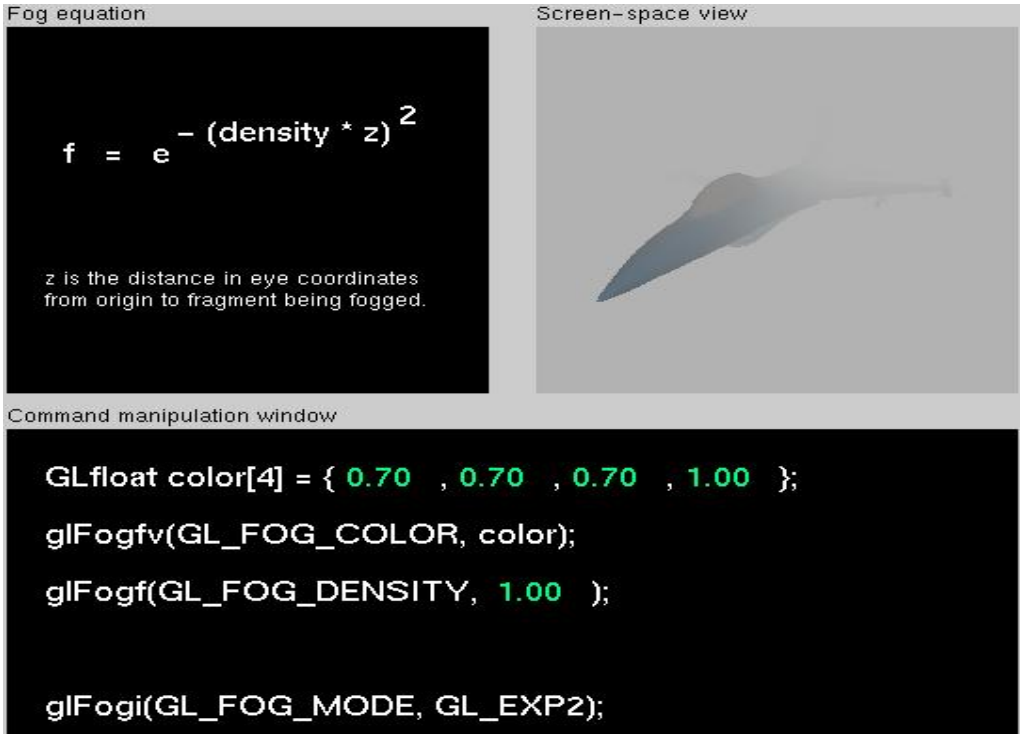
مثال ۱:


Fog equation
f = end – z / end – start
z is the distance in eye coordinates from origin to fragment being fogged.
Screen-space view
Command manipulation window
GLfloat color[4] = { 0.70 , 0.70 , 0.70 , 1.00 };
glFogfv(GL_FOG_COLOR, color);
glFogf(GL_FOG_START, 0.50);
glFogf(GL_FOG_END, 2.00);
glFogi(GL_FOG_MODE, GL_LINEAR);

مثال۲:



مثال٣:



والآن سنعرض مثال كامل لطريقة استخدام الضباب في نظام الألوان RGBA .

مثال١:

سنرسم في التطبيق الرسومي التالي خمسة أباريق تخضع لتأثير الضباب، ولكن تختلف درجة كثافة الضباب من أجل كل أبريق كما يمكن تغيير تابع الكثافة عن طريق التحكم بلوحة المفاتيح .

سيكون خرج التطبيق كما في الشكل التالي :

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>
#include "aux.h"

GLint fogMode;

void cycleFog (AUX_EVENTREC *event)
{
    if (fogMode == GL_EXP) {
        fogMode = GL_EXP2;
        printf ("Fog mode is GL_EXP2\n");
    }
    else if (fogMode == GL_EXP2) {
        fogMode = GL_LINEAR;
        printf ("Fog mode is GL_LINEAR\n");
        glFogf (GL_FOG_START, 1.0);
        glFogf (GL_FOG_END, 5.0);
    }
    else if (fogMode == GL_LINEAR) {
        fogMode = GL_EXP;
        printf ("Fog mode is GL_EXP\n");
    }
    glFogi (GL_FOG_MODE, fogMode);
}

void myinit(void)
{
    GLfloat position[] = { 0.0, 3.0, 3.0, 0.0 };
    GLfloat local_view[] = { 0.0 };

    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glLightfv(GL_LIGHT0, GL_POSITION, position);
    glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, local_view);

    glFrontFace (GL_CW);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_AUTO_NORMAL);
    glEnable(GL_NORMALIZE);
    glEnable(GL_FOG);
    {
        GLfloat density;
        GLfloat fogColor[4] = {0.5, 0.5, 0.5, 1.0};

        fogMode = GL_EXP;
        glFogi (GL_FOG_MODE, fogMode);
        glFogfv (GL_FOG_COLOR, fogColor);
        glFogf (GL_FOG_DENSITY, 0.35);
        glHint (GL_FOG_HINT, GL_DONT_CARE);
        glClearColor(0.5, 0.5, 0.5, 1.0);
    }
}
```

```
void renderRedTeapot (GLfloat x, GLfloat y, GLfloat z)
{
    float mat[3];

    glPushMatrix();
    glTranslatef (x, y, z);
    mat[0] = 0.1745; mat[1] = 0.01175; mat[2] = 0.01175;
    glMaterialfv (GL_FRONT, GL_AMBIENT, mat);
    mat[0] = 0.61424; mat[1] = 0.04136; mat[2] = 0.04136;
    glMaterialfv (GL_FRONT, GL_DIFFUSE, mat);
    mat[0] = 0.727811; mat[1] = 0.626959; mat[2] = 0.626959;
    glMaterialfv (GL_FRONT, GL_SPECULAR, mat);
    glMaterialf (GL_FRONT, GL_SHININESS, 0.6*128.0);
    auxSolidTeapot(1.0);
    glPopMatrix();
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    renderRedTeapot (-4.0, -0.5, -1.0);
    renderRedTeapot (-2.0, -0.5, -2.0);
    renderRedTeapot (0.0, -0.5, -3.0);
    renderRedTeapot (2.0, -0.5, -4.0);
    renderRedTeapot (4.0, -0.5, -5.0);
    glFlush();
}

void myReshape(GLsizei w, GLsizei h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= (h*3))
        glOrtho (-6.0, 6.0, -2.0*((GLfloat) h*3)/(GLfloat) w,
         2.0*((GLfloat) h*3)/(GLfloat) w, 0.0, 10.0);
    else
        glOrtho (-6.0*(GLfloat) w/((GLfloat) h*3),
         6.0*(GLfloat) w/((GLfloat) h*3), -2.0, 2.0, 0.0,
10.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity ();
}
int main(int argc, char** argv)
{
    auxInitDisplayMode (AUX_SINGLE | AUX_RGBA | AUX_DEPTH);
    auxInitPosition (0, 0, 450, 150);
    auxInitWindow (argv[0]);
    auxMouseFunc (AUX_LEFTBUTTON, AUX_MOUSEDOWN, cycleFog);
    myinit();
    auxReshapeFunc (myReshape);
    auxMainLoop(display);
}
```

مثال٢:

يوضح هذا المثال استخدام الضباب في نظام الألوان color-index :

```
include <GL/gl.h>
#include <GL/glu.h>
#include "aux.h"

#define NUMCOLORS 32
#define RAMPSTART 16

void myinit(void)
{
    int i;

    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    for (i = 0; i < NUMCOLORS; i++) {
        GLfloat shade;
        shade = (GLfloat) (NUMCOLORS-i)/(GLfloat) NUMCOLORS;
        auxSetOneColor (16 + i, shade, shade, shade);
    }
    glEnable(GL_FOG);

    glFogi (GL_FOG_MODE, GL_LINEAR);
    glFogi (GL_FOG_INDEX, NUMCOLORS);
    glFogf (GL_FOG_START, 0.0);
    glFogf (GL_FOG_END, 4.0);
    glHint (GL_FOG_HINT, GL_NICEST);
    glClearIndex((GLfloat) (NUMCOLORS+RAMPSTART-1));
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix ();
        glTranslatef (-1.0, -1.0, -1.0);
        glRotatef (-90.0, 1.0, 0.0, 0.0);
        glIndexi (RAMPSTART);
        auxSolidCone(1.0, 2.0);
    glPopMatrix ();

    glPushMatrix ();
        glTranslatef (0.0, -1.0, -2.25);
        glRotatef (-90.0, 1.0, 0.0, 0.0);
        glIndexi (RAMPSTART);
        auxSolidCone(1.0, 2.0);
    glPopMatrix ();
    glPushMatrix ();
        glTranslatef (1.0, -1.0, -3.5);
        glRotatef (-90.0, 1.0, 0.0, 0.0);
        glIndexi (RAMPSTART);
        auxSolidCone(1.0, 2.0);
    glPopMatrix ();
    glFlush();
}
```

```
void myReshape(GLsizei w, GLsizei h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        glOrtho (-2.0, 2.0, -2.0*(GLfloat)h/(GLfloat)w,
            2.0*(GLfloat)h/(GLfloat)w, 0.0, 10.0);
    else
        glOrtho (-2.0*(GLfloat)w/(GLfloat)h,
            2.0*(GLfloat)w/(GLfloat)h, -2.0, 2.0, 0.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity ();
}


int main(int argc, char** argv)
{
    auxInitDisplayMode (AUX_SINGLE | AUX_INDEX | AUX_DEPTH);
    auxInitPosition (0, 0, 200, 200);
    auxInitWindow (argv[0]);
    myinit();
    auxReshapeFunc (myReshape);
    auxMainLoop(display);
}
```

الفصل السابع
مواضيع متنوعة

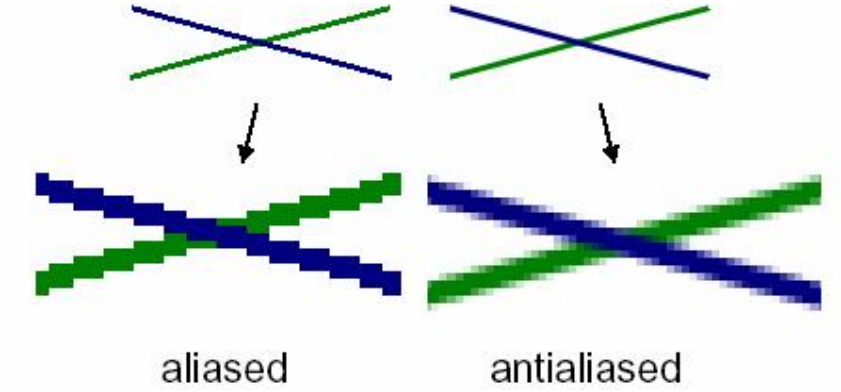
مقدمة:

سنعرض في هذا الفصل بعض المواضيع المتنوعة، والتي يمكن تحقيقها باستخدام تعليمات الـــ OpenGL.

تنعيم الخطوط Antialiasing:

غالبا ما تظهر الرسوم في الحاسب بحواف خشنة ومتكسرة، حيث كلنا يعلم أن الخشونة تنشأ أصلاً عن الدقة المنخفضة للإظهار لذلك فإن أول عامل في إزلة الخشونة هو رفع دقة الإظهار.

انظر الشكل التالي :



يظهر هنا بشكل واضح وجلي الاختلاف بين الشكلين من حيث نعومة وتكسر الخطوط.

وبما أن الـــ OpenGL تسعى إلى إظهار أفضل وأدق النماذج الرسومية كان لابد لها أن تهتم بهذا الأمـــر، فقدمت تعليمات خاصة تساعد على التخلص من مشكلة الخطوط المتكسرة .

أولاً:

```
void glHint(GLenum target, GLenum hint);
```

حيث تحقق هذه التعليمة فكرة تنعيم الخطوط ولكن وفق معايير يتم تحديدها بالوسطاء الممرة لها، حيث تهتم هذه المعايير بالدقة الكبيرة في تنعيم ومعالجة الخطوط أو بالسرعة في تنفيذ عملية التنعيم. مع العلم أنه توجد تطبيقات لا تتأثر بالـــ **glHint.**

الوسطاء:

hint: يأخذ هذا الوسيط إحدى القيم التالية:

- **GL_FASTEST:** للإشارة إلى الاهتمام بسرعة المعالجة.
- **GL_NICEST:** للإشارة إلى الاهتمام بدقة التنفيذ.
- **GL_DONT_CARE:** للإشارة إلى عدم وجود أولوية لمعيار على الآخر.

أما الوسيط **target** فهو يأخذ إحدى القيم الموضحة في الجدول التالي:

الوسيط	المعنى
GL_POINT_SMOOTH_HINT, GL_LINE_SMOOTH_HINT, GL_POLYGON_SMOOTH_HINT	تحديد عملية التنعيم على ماذا ستنفذ نقاط أو خطوط أو مضلعات
GL_FOG_HINT	خاص بالضباب: تحيدي فيما إذا كان الضباب سينفذ من أجل كل بكسل (GL_NICEST) أو كل عقدة (GL_FASTEST)
GL_PERSPECTIVE_CORRECTION_HINT	Specifies the desired quality of color and texture-coordinate interpolation

لتحقيق عملية التنعيم على النقاط والخطوط والمضلع نستخدم التعليمة :

glEnable( )

مع وسيط يأخذ إحدى القيم التالية:

GL_POINT_SMOOTH ،GL_LINE_SMOOTHK ، GL_POLYGON_SMOOTHK

مثال:

```
glEnable (GL_LINE_SMOOTH);// enable the antialiasing for lines
glHint (GL_LINE_SMOOTH_HINT, GL_NICEST); // for best quality
glEnable (GL_BLEND);
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glLineWidth (3.0); // set the width of line to 3 pixels
glBegin (GL_LINES);
     glVertex2f (-0.5, 0.5);
     glVertex2f (0.5, -0.5);
glEnd ();
```

لتحقيق عملية التنعيم في نظام الألوان الـــ RGBA يجب تفعيل الـــ Blending بالشكل التالي:

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

مثال:

Example: An Antialiased Wireframe Icosahedron: anti.c

```
#include <GL/gl.h>
#include <GL/glu.h>
#include "aux.h"

void myinit(void)
{
    GLfloat values[2];
    glGetFloatv(GL_LINE_WIDTH_GRANULARITY, values);
    printf("GL_LINE_WIDTH_GRANULARITY value is %3.1f\n",
                values[0]);
    glGetFloatv(GL_LINE_WIDTH_RANGE, values);
    printf("GL_LINE_WIDTH_RANGE values are %3.1f %3.1f\n",
                values[0], values[1]);
    glEnable(GL_LINE_SMOOTH);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glHint(GL_LINE_SMOOTH_HINT, GL_DONT_CARE);
    glLineWidth(1.5);
    glShadeModel(GL_FLAT);
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glDepthFunc(GL_LEQUAL);
    glEnable(GL_DEPTH_TEST);
}
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor4f(1.0, 1.0, 1.0, 1.0);
    auxWireIcosahedron(1.0);
    glFlush();
}
void myReshape(GLsizei w, GLsizei h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (GLfloat) w/(GLfloat) h, 3.0, 5.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0, 0.0, -4.0);
}
int main(int argc, char** argv)
{
    auxInitDisplayMode(AUX_SINGLE | AUX_RGBA | AUX_DEPTH);
    auxInitPosition(0, 0, 400, 400);
    auxInitWindow(argv[0]);
    myinit();
    auxReshapeFunc(myReshape);
    auxMainLoop(display);
}
```

مثال يضم حوالي ٩٩% من تعليمات مكتبة الـــ GLUT.

```
**
 * My first GLUT prog.
 * Uses most GLUT calls to prove it all works.
 *
 * Notes:
 * Display lists are not shared between windows, and there doesn't
seem to be
 *  any provision for this in GLUT. See glxCreateContext.
 *
 * The windows are internally indexed 0,1,2,3,4,5,6. The actual
window ids
 * returned by glut are held in winId[0], ...
 *
 * Todo:
 *
 * Could reorder the windows so 0,1,2,3,4,5,6 are the gfx, 7,8 text.
 *
 * Calls not used yet: these callbacks are registered but inactive.
 *
 *  glutSpaceball<xxx>Func
 *  glutButtonBoxFunc
 *  glutDialsFunc
 *  glutTabletMotionFunc
 *  glutTabletButtonFunc
 *
 * Tested on:
 *  R3K Indigo Starter
 *  R4K Indigo Elan
 *  R4K Indy XZ
 *  R4K Indy XL
 *  R4K Indigo2 Extreme
 */

#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

/* Controls */

#define VERSION    "1.00"
#define DATE       "16Sep95"
#define DELAY      1000 /* delay for timer test */
#define MENUDELAY  200  /* hack to fix glutMenuStateFunc bug */
#define MAXWIN     9    /* max no. of windows */

unsigned int AUTODELAY = 1500;   /* delay in demo mode  */

#define VERSIONLONG "Version " VERSION "/" DATE ", compiled "
__DATE__ ", " __TIME__ ", file " __FILE__

int pos[MAXWIN][2] =
{
  {50, 150},            /* win 0  */
  {450, 150},           /* win 1  */
```

```
  {50, 600},            /* win 2  */
  {450, 600},           /* win 3  */
  {10, 10},             /* subwin 4 (relative to parent win 0) */
  {300, 400},           /* help win 5  */
  {850, 150},           /* cmap win 6  */
  {850, 600},           /* cmap win 7  */
  {250, 450}            /* text win 8  */
};

int size[MAXWIN][2] =
{
  {350, 350},           /* win 0  */
  {350, 350},           /* win 1  */
  {350, 350},           /* win 2  */
  {350, 350},           /* win 3  */
  {200, 200},           /* subwin 4  */
  {700, 300},           /* help win 5  */
  {350, 350},           /* cmap win 6  */
  {350, 350},           /* cmap win 7  */
  {800, 450}            /* text win 8  */
};

/* Macros */

#define PR     if(debug)printf

/* #define GLNEWLIST(a, b)  glNewList(a, b), fprintf(stderr,
   "creating list %d \n", a); */
/* #define GLCALLLIST(a)    glCallList(a), fprintf(stderr,
   "calling list %d \n", a); */
/* #define GLUTSETWINDOW(x) glutSetWindow(x), fprintf(stderr,
   "gsw at %d\n", __LINE__) */

/* Globals */

int winId[MAXWIN] =
{0};                    /* table of glut window id's  */
GLboolean winVis[MAXWIN] =
{GL_FALSE};                /* is window visible  */

GLboolean text[MAXWIN] =
{GL_FALSE};                /* is text on  */
GLboolean winFreeze[MAXWIN] =
{GL_FALSE};                /* user requested menuFreeze  */
GLboolean menuFreeze = GL_FALSE;  /* menuFreeze while menus posted
*/
GLboolean timerOn = GL_FALSE;  /* timer active  */
GLboolean animation = GL_TRUE;  /* idle func animation on  */
GLboolean debug = GL_FALSE;  /* dump all events  */
GLboolean showKeys = GL_FALSE;  /* dump key events  */
GLboolean demoMode = GL_FALSE;  /* run automatic demo  */
GLboolean backdrop = GL_FALSE;  /* use backdrop polygon  */
GLboolean passive = GL_FALSE;  /* report passive motions  */
GLboolean leftDown = GL_FALSE;  /* left button down ?  */
GLboolean middleDown = GL_FALSE;  /* middle button down ?  */

int displayMode = GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH;
int currentShape = 0;   /* current glut shape  */
int scrollLine = 0, scrollCol = 0;  /* help scrolling params  */
int lineWidth = 1;      /* line width  */
int angle = 0;      /* global rotation angle  */
```

```
char *textPtr[1000] =
{0};                    /* pointers to text window text  */
int textCount = 0, helpCount = 0;  /* text list indexes  */
float scaleFactor = 0.0;  /* window size scale factor  */

#ifdef ALPHA
#undef ALPHA  /* Avoid problems with DEC's ALPHA machines. */
#endif

int menu1, menu2, menu3, menu4, menu5, menu6 = 0, menu7, menu8;
enum {
  RGBA, INDEX, SINGLE, DOUBLEBUFFER, DEPTH, ACCUM, ALPHA, STENCIL,
MULTISAMPLE,
  STEREO, MODES
};
char *modeNames[] =
{"RGBA", "INDEX", "SINGLE", "DOUBLE", "DEPTH", "ACCUM",
  "ALPHA", "STENCIL", "MULTISAMPLE", "STEREO"};
int glutMode[] =
{GLUT_RGBA, GLUT_INDEX, GLUT_SINGLE, GLUT_DOUBLE, GLUT_DEPTH,
  GLUT_ACCUM, GLUT_ALPHA, GLUT_STENCIL, GLUT_MULTISAMPLE,
GLUT_STEREO};
int modes[MODES] =
{0};
GLboolean menuButton[3] =
{0, 0, 1};
enum {
  MOUSEBUTTON, MOUSEMOTION, APPLY, RESET
};

/* Prototypes */

void gfxInit(int);
void drawScene(void);
void idleFunc(void);
void reshapeFunc(int width, int height);
void visible(int state);
void keyFunc(unsigned char key, int x, int y);
void mouseFunc(int button, int state, int x, int y);
void motionFunc(int x, int y);
void passiveMotionFunc(int x, int y);
void entryFunc(int state);
void specialFunc(int key, int x, int y);
void menuStateFunc(int state);
void timerFunc(int value);
#if 0
void delayedReinstateMenuStateCallback(int value);
#endif
void menuFunc(int value);
void showText(void);
void textString(int x, int y, char *str, void *font);
void strokeString(int x, int y, char *str, void *font);
void makeMenus(void);
int idToIndex(int id);
void setInitDisplayMode(void);
void createMenu6(void);
void removeCallbacks(void);
void addCallbacks(void);
void dumpIds(void);
void updateHelp(void);
void updateAll(void);
```

```
void killWindow(int index);
void makeWindow(int index);
void warning(char *msg);
void autoDemo(int);
void positionWindow(int index);
void reshapeWindow(int index);
void iconifyWindow(int index);
void showWindow(int index);
void hideWindow(int index);
void pushWindow(int index);
void popWindow(int index);
void attachMenus(void);
void killAllWindows(void);
void makeAllWindows(void);
void updateText(void);
void updateScrollWindow(int index, char **ptr);
void redefineShapes(int shape);
GLboolean match(char *, char *);
void checkArgs(int argc, char *argv[]);
void scaleWindows(float);
void commandLineHelp(void);
void trackBall(int mode, int button, int state, int x, int y);

void spaceballMotionCB(int, int, int);
void spaceballRotateCB(int, int, int);
void spaceballButtonCB(int, int);
void buttonBoxCB(int, int);
void dialsCB(int, int);
void tabletMotionCB(int, int);
void tabletButtonCB(int, int, int, int);

/* strdup is actually not a standard ANSI C or POSIX routine
   so implement a private one.  OpenVMS does not have a strdup;
Linux's
   standard libc doesn't declare strdup by default (unless BSD or
SVID
   interfaces are requested). */
static char *
stralloc(const char *string)
{
  char *copy;

  copy = malloc(strlen(string) + 1);
  if (copy == NULL)
    return NULL;
  strcpy(copy, string);
  return copy;
}

/* main */

int
main(int argc, char **argv)
{

/* General init */

  glutInit(&argc, argv);

/* Check args */
```

```
  checkArgs(argc, argv);

/* Scale window position/size if needed. Ignore aspect ratios. */

  if (scaleFactor > 0.0)
    scaleWindows(scaleFactor);
  else
    scaleWindows(glutGet(GLUT_SCREEN_WIDTH) / 1280.0);

/* Set initial display mode */

  modes[RGBA] = 1;
  modes[DOUBLEBUFFER] = 1;
  modes[DEPTH] = 1;
  setInitDisplayMode();

/* Set up menus */

  makeMenus();

/* Make some windows */

  makeWindow(0);
  makeWindow(1);

/* Global callbacks */

  glutIdleFunc(idleFunc);
  glutMenuStateFunc(menuStateFunc);

/* Start demo if needed */

  if (demoMode)
    autoDemo(-2);

/* Fall into event loop */

  glutMainLoop();
  return 0;             /* ANSI C requires main to return int. */
}

/* gfxInit - Init opengl for each window */

void
gfxInit(int index)
{
  GLfloat grey10[] =
  {0.10, 0.10, 0.10, 1.0};
  GLfloat grey20[] =
  {0.2, 0.2, 0.2, 1.0};
  GLfloat black[] =
  {0.0, 0.0, 0.0, 0.0};
  GLfloat diffuse0[] =
  {1.0, 0.0, 0.0, 1.0};
  GLfloat diffuse1[] =
  {0.0, 1.0, 0.0, 1.0};
  GLfloat diffuse2[] =
  {1.0, 1.0, 0.0, 1.0};
  GLfloat diffuse3[] =
  {0.0, 1.0, 1.0, 1.0};
  GLfloat diffuse4[] =
```

```
  {1.0, 0.0, 1.0, 1.0};

#define XX  3
#define YY  3
#define ZZ  -2.5

  float vertex[][3] =
  {
    {-XX, -YY, ZZ},
    {+XX, -YY, ZZ},
    {+XX, +YY, ZZ},
    {-XX, +YY, ZZ}
  };

/* warning: This func mixes RGBA and CMAP calls in an ugly
   fashion */

  redefineShapes(currentShape);  /* set up display lists  */
  glutSetWindow(winId[index]);  /* hack - redefineShapes
                                   changes glut win */

/* Shaded backdrop square (RGB or CMAP) */

  glNewList(100, GL_COMPILE);
  glPushAttrib(GL_LIGHTING);
  glDisable(GL_LIGHTING);
  glBegin(GL_POLYGON);

  glColor4fv(black);
  glIndexi(0);
  glVertex3fv(vertex[0]);

  glColor4fv(grey10);
  glIndexi(3);
  glVertex3fv(vertex[1]);

  glColor4fv(grey20);
  glIndexi(4);
  glVertex3fv(vertex[2]);

  glColor4fv(grey10);
  glIndexi(7);
  glVertex3fv(vertex[3]);

  glEnd();
  glPopAttrib();
  glIndexi(9);
  glEndList();

/* Set proj+view */

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(40.0, 1.0, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.);
  glTranslatef(0.0, 0.0, -1.0);

  if (index == 6 || index == 7)
    goto colorindex;
```

```
/* Set basic material, lighting for RGB windows */

  if (index == 0)
    glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse0);
  else if (index == 1)
    glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse1);
  else if (index == 2)
    glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse2);
  else if (index == 3)
    glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse3);
  else if (index == 4)
    glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse4);

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glEnable(GL_DEPTH_TEST);

  if (index == 4)
    glClearColor(0.15, 0.15, 0.15, 1);
  else
    glClearColor(0.1, 0.1, 0.1, 1.0);

  return;

/* Set GL basics for CMAP windows 6,7 */

colorindex:

  glEnable(GL_DEPTH_TEST);
  if (glutGet(GLUT_WINDOW_COLORMAP_SIZE) < 16)
    warning("Color map size too small for color index window");

/* Try to reuse an existing color map */

  if ((index == 6) && (winId[7] != 0)) {
    glutCopyColormap(winId[7]);
  } else if ((index == 7) && (winId[6] != 0)) {
    glutCopyColormap(winId[6]);
  } else {
    glutSetColor(8, 0.1, 0.1, 0.1);
    glutSetColor(9, 1.0, 0.5, 0.0);
    glutSetColor(10, 1.0, 0.6, 0.8);
  }
  glClearIndex(8);
  glIndexi(index + 3);

}

/* makeMenus - Create popup menus */

void
makeMenus(void)
{

/* General control / debug */

  menu2 = glutCreateMenu(menuFunc);
  glutAddMenuEntry("toggle auto demo mode (a)", 312);
  glutAddMenuEntry("toggle freezing in menus", 300);
  glutAddMenuEntry("toggle text per window (t)", 301);
```

```
  glutAddMenuEntry("toggle global timer", 302);
  glutAddMenuEntry("toggle global animation", 303);
  glutAddMenuEntry("toggle per window animation", 304);
  glutAddMenuEntry("toggle debug prints (D)", 305);
  glutAddMenuEntry("toggle shaded backdrop", 307);
  glutAddMenuEntry("toggle passive motion callback", 308);
  glutAddMenuEntry("increase line width (l)", 310);
  glutAddMenuEntry("decrease line width  (L)", 311);

/* Shapes */

  menu3 = glutCreateMenu(menuFunc);
  glutAddMenuEntry("sphere", 200);
  glutAddMenuEntry("cube", 201);
  glutAddMenuEntry("cone", 202);
  glutAddMenuEntry("torus", 203);
  glutAddMenuEntry("dodecahedron", 204);
  glutAddMenuEntry("octahedron", 205);
  glutAddMenuEntry("tetrahedron", 206);
  glutAddMenuEntry("icosahedron", 207);
  glutAddMenuEntry("teapot", 208);

/* Open/close windows */

  menu4 = glutCreateMenu(menuFunc);
  glutAddMenuEntry("open all windows", 450);
  glutAddMenuEntry("close all windows", 451);
  glutAddMenuEntry(" ", 9999);
  glutAddMenuEntry("create win 0", 400);
  glutAddMenuEntry("create win 1", 401);
  glutAddMenuEntry("create win 2", 402);
  glutAddMenuEntry("create win 3", 403);
  glutAddMenuEntry("create sub window", 404);
  glutAddMenuEntry("create color index win 6", 406);
  glutAddMenuEntry("create color index win 7", 407);
  glutAddMenuEntry(" ", 9999);
  glutAddMenuEntry("destroy win 0", 410);
  glutAddMenuEntry("destroy win 1", 411);
  glutAddMenuEntry("destroy win 2", 412);
  glutAddMenuEntry("destroy win 3", 413);
  glutAddMenuEntry("destroy sub window", 414);
  glutAddMenuEntry("destroy color index win 6", 416);
  glutAddMenuEntry("destroy color index win 7", 417);

/* Window manager stuff */

  menu5 = glutCreateMenu(menuFunc);
  glutAddMenuEntry("move current win", 430);
  glutAddMenuEntry("resize current win", 431);
  glutAddMenuEntry("iconify current win", 432);
  glutAddMenuEntry("show current win", 433);
  glutAddMenuEntry("hide current win", 434);
  glutAddMenuEntry("push current win", 435);
  glutAddMenuEntry("pop current win", 436);
  glutAddMenuEntry(" ", 9999);
  glutAddMenuEntry("move win 1", 420);
  glutAddMenuEntry("resize win 1", 421);
  glutAddMenuEntry("iconify win 1", 422);
  glutAddMenuEntry("show win 1", 423);
  glutAddMenuEntry("hide win 1", 424);
  glutAddMenuEntry("push win 1", 425);
```

```
  glutAddMenuEntry("pop win 1", 426);

/* Gfx modes */

  createMenu6();        /* build dynamically  */

/* Texty reports */

  menu7 = glutCreateMenu(menuFunc);
  glutAddMenuEntry("report current win modes", 700);
  glutAddMenuEntry("report current device data", 701);
  glutAddMenuEntry("check OpenGL extensions", 702);
  glutAddMenuEntry("dump internal data (d)", 703);

/* Play with menus */

  menu8 = glutCreateMenu(menuFunc);
  glutAddMenuEntry("toggle menus on left button", 805);
  glutAddMenuEntry("toggle menus on middle button", 806);
  glutAddMenuEntry("toggle menus on right button", 807);
  glutAddMenuEntry("--------------------------", 9999);
  glutAddMenuEntry("add plain items", 800);
  glutAddMenuEntry("add submenu items", 801);
  glutAddMenuEntry("change new entries to plain items", 802);
  glutAddMenuEntry("change new entries to submenus", 803);
  glutAddMenuEntry("remove all new items", 804);
  glutAddMenuEntry("--------------------------", 9999);

/* Main menu */

  menu1 = glutCreateMenu(menuFunc);
  glutAddSubMenu("control", menu2);
  glutAddSubMenu("shapes", menu3);
  glutAddSubMenu("windows", menu4);
  glutAddSubMenu("window ops", menu5);
  glutAddSubMenu("gfx modes", menu6);
  glutAddSubMenu("reports", menu7);
  glutAddSubMenu("menus", menu8);
  glutAddMenuEntry("help (h)", 101);
  glutAddMenuEntry("quit (esc)", 100);
}

/* createMenu6 - Dynamically rebuild menu of display modes to
   show current choices */

void
createMenu6(void)
{
  char str[100];
  int i;

  if (menu6 != 0)
    glutDestroyMenu(menu6);
  menu6 = glutCreateMenu(menuFunc);

  for (i = 0; i < MODES; i++) {
    sprintf(str, "%srequest %s", (modes[i] ? "+ " : "   "),
modeNames[i]);
    glutAddMenuEntry(str, 602 + i);
  }
}
```

```
/* menuFunc - Process return codes from popup menus */

void
menuFunc(int value)
{
  static int initItems = 10;
  int items, m;

  if (initItems == 0) {
    glutSetMenu(menu8);
    initItems = glutGet(GLUT_MENU_NUM_ITEMS);
  }
  PR("Menu returned value %d \n", value);

  switch (value) {

/* GLUT shapes */

  case 200:
  case 201:
  case 202:
  case 203:
  case 204:
  case 205:
  case 206:
  case 207:
  case 208:
    redefineShapes(value - 200);
    break;

/* Overall controls */

  case 300:
    menuFreeze = !menuFreeze;
    break;

  case 301:
    text[idToIndex(glutGetWindow())] =
!(text[idToIndex(glutGetWindow())]);
    break;

  case 302:
    timerOn = !timerOn;
    if (timerOn)
      glutTimerFunc(DELAY, timerFunc, 1);
    break;

  case 303:
    animation = !animation;
    if (animation)
      glutIdleFunc(idleFunc);
    else
      glutIdleFunc(NULL);
    break;

  case 304:
    winFreeze[idToIndex(glutGetWindow())] = !(winFreeze[idToIndex(
          glutGetWindow())]);
    break;
```

```
  case 305:
    debug = !debug;
    break;

  case 307:
    backdrop = !backdrop;
    break;

  case 308:
    passive = !passive;
    if (passive)
      glutPassiveMotionFunc(passiveMotionFunc);
    else
      glutPassiveMotionFunc(NULL);
    break;

  case 310:
    lineWidth += 1;
    updateAll();
    break;

  case 311:
    lineWidth -= 1;
    if (lineWidth < 1)
      lineWidth = 1;
    updateAll();
    break;

  case 312:
    demoMode = !demoMode;
    if (demoMode)
      autoDemo(-2);
    break;

/* Window create/destroy. */

/* Creates */

  case 400:
    makeWindow(0);
    break;

  case 401:
    makeWindow(1);
    break;

  case 402:
    makeWindow(2);
    break;

  case 403:
    makeWindow(3);
    break;

  case 404:
    makeWindow(4);
    break;

  case 406:
    makeWindow(6);
    break;
```

```
  case 407:
    makeWindow(7);
    break;

/* Destroys */

  case 410:
    killWindow(0);
    break;

  case 411:
    killWindow(1);
    break;

  case 412:
    killWindow(2);
    break;

  case 413:
    killWindow(3);
    break;

  case 414:
    killWindow(4);
    break;

  case 416:
    killWindow(6);
    break;

  case 417:
    killWindow(7);
    break;

  case 450:
    makeAllWindows();
    break;

  case 451:
    killAllWindows();
    break;

/* Window movements etc. */

  case 420:
    positionWindow(1);
    break;

  case 421:
    reshapeWindow(1);
    break;

  case 422:
    iconifyWindow(1);
    break;

  case 423:
    showWindow(1);
    break;
```

```
  case 424:
    hideWindow(1);
    break;

  case 425:
    pushWindow(1);
    break;

  case 426:
    popWindow(1);
    break;

  case 430:
    positionWindow(idToIndex(glutGetWindow()));
    break;

  case 431:
    reshapeWindow(idToIndex(glutGetWindow()));
    break;

  case 432:
    iconifyWindow(idToIndex(glutGetWindow()));
    break;

  case 433:
    showWindow(idToIndex(glutGetWindow()));
    break;

  case 434:
    hideWindow(idToIndex(glutGetWindow()));
    break;

  case 435:
    pushWindow(idToIndex(glutGetWindow()));
    break;

  case 436:
    popWindow(idToIndex(glutGetWindow()));
    break;

/* Test gfx modes. */

  case 600:
    makeWindow(3);
    break;

  case 601:
    killWindow(3);
    break;

  case 602:
  case 603:
  case 604:
  case 605:
  case 606:
  case 607:
  case 608:
  case 609:
  case 610:
  case 611:
    modes[value - 602] = !modes[value - 602];
```

```
    setInitDisplayMode();
    break;

/* Text reports */

/* This is pretty ugly. */

#define INDENT 30
#define REPORTSTART(text)                          \
        printf("\n" text "\n");                    \
        textPtr[0] = (char *)malloc(strlen(text)+1); \
        strcpy(textPtr[0], text);                  \
        textCount = 1;

#define REPORTEND                                  \
        scrollLine = 0;                            \
        textPtr[textCount] = NULL;                 \
        makeWindow(8);                             \
        updateText();

#define GLUTGET(name)                              \
       {                                           \
          char str[100], str2[100];                \
          int s, len;                              \
          sprintf(str, # name);                    \
          len = (int) strlen(# name);              \
          for(s = 0 ; s < INDENT-len; s++)         \
            strcat(str, " ");                      \
          sprintf(str2, ": %d\n",glutGet(name));   \
          strcat(str, str2);                       \
          printf(str);                             \
          textPtr[textCount] = stralloc(str);      \
          textCount++;                             \
       }

  case 700:

    printf("XXXXXX glutGetWindow = %d\n", glutGetWindow());
    REPORTSTART("glutGet():");

    GLUTGET(GLUT_WINDOW_X);
    GLUTGET(GLUT_WINDOW_Y);
    GLUTGET(GLUT_WINDOW_WIDTH);
    GLUTGET(GLUT_WINDOW_HEIGHT);
    GLUTGET(GLUT_WINDOW_BUFFER_SIZE);
    GLUTGET(GLUT_WINDOW_STENCIL_SIZE);
    GLUTGET(GLUT_WINDOW_DEPTH_SIZE);
    GLUTGET(GLUT_WINDOW_RED_SIZE);
    GLUTGET(GLUT_WINDOW_GREEN_SIZE);
    GLUTGET(GLUT_WINDOW_BLUE_SIZE);
    GLUTGET(GLUT_WINDOW_ALPHA_SIZE);
    GLUTGET(GLUT_WINDOW_ACCUM_RED_SIZE);
    GLUTGET(GLUT_WINDOW_ACCUM_GREEN_SIZE);
    GLUTGET(GLUT_WINDOW_ACCUM_BLUE_SIZE);
    GLUTGET(GLUT_WINDOW_ACCUM_ALPHA_SIZE);
    GLUTGET(GLUT_WINDOW_DOUBLEBUFFER);
    GLUTGET(GLUT_WINDOW_RGBA);
    GLUTGET(GLUT_WINDOW_PARENT);
    GLUTGET(GLUT_WINDOW_NUM_CHILDREN);
    GLUTGET(GLUT_WINDOW_COLORMAP_SIZE);
    GLUTGET(GLUT_WINDOW_NUM_SAMPLES);
```

```
    GLUTGET(GLUT_STEREO);
    GLUTGET(GLUT_SCREEN_WIDTH);
    GLUTGET(GLUT_SCREEN_HEIGHT);
    GLUTGET(GLUT_SCREEN_HEIGHT_MM);
    GLUTGET(GLUT_SCREEN_WIDTH_MM);
    GLUTGET(GLUT_MENU_NUM_ITEMS);
    GLUTGET(GLUT_DISPLAY_MODE_POSSIBLE);
    GLUTGET(GLUT_INIT_DISPLAY_MODE);
    GLUTGET(GLUT_INIT_WINDOW_X);
    GLUTGET(GLUT_INIT_WINDOW_Y);
    GLUTGET(GLUT_INIT_WINDOW_WIDTH);
    GLUTGET(GLUT_INIT_WINDOW_HEIGHT);
    GLUTGET(GLUT_ELAPSED_TIME);

    REPORTEND;
    break;

#define GLUTDEVGET(name)                          \
        {                                         \
          char str[100], str2[100];               \
          int len, s;                             \
          sprintf(str, # name);                   \
          len = (int) strlen(# name);             \
          for(s = 0 ; s < INDENT-len; s++)        \
            strcat(str, " ");                     \
          sprintf(str2, ": %d\n",                 \
             glutDeviceGet(name));                \
          strcat(str, str2);                      \
          printf(str);                            \
          textPtr[textCount] = stralloc(str);     \
          textCount++;                            \
        }

  case 701:
    REPORTSTART("glutDeviceGet():");

    GLUTDEVGET(GLUT_HAS_KEYBOARD);
    GLUTDEVGET(GLUT_HAS_MOUSE);
    GLUTDEVGET(GLUT_HAS_SPACEBALL);
    GLUTDEVGET(GLUT_HAS_DIAL_AND_BUTTON_BOX);
    GLUTDEVGET(GLUT_HAS_TABLET);
    GLUTDEVGET(GLUT_NUM_MOUSE_BUTTONS);
    GLUTDEVGET(GLUT_NUM_SPACEBALL_BUTTONS);
    GLUTDEVGET(GLUT_NUM_BUTTON_BOX_BUTTONS);
    GLUTDEVGET(GLUT_NUM_DIALS);
    GLUTDEVGET(GLUT_NUM_TABLET_BUTTONS);

    REPORTEND;
    break;

#define EXTCHECK(name)                            \
        {                                         \
          char str[100], str2[100];               \
          int len, s;                             \
          sprintf(str, # name);                   \
          len = (int) strlen(# name);             \
          for(s = 0 ; s < INDENT-len; s++)        \
            strcat(str, " ");                     \
          sprintf(str2, ": %s\n",                 \
             glutExtensionSupported(# name)?      \
               "yes": "no");                      \
```

```
          strcat(str, str2);                    \
          printf(str);                          \
          textPtr[textCount] = stralloc(str);   \
          textCount++;                          \
        }

  case 702:
    REPORTSTART("glutExtensionSupported():");

    EXTCHECK(GL_EXT_abgr);
    EXTCHECK(GL_EXT_blend_color);
    EXTCHECK(GL_EXT_blend_minmax);
    EXTCHECK(GL_EXT_blend_logic_op);
    EXTCHECK(GL_EXT_blend_subtract);
    EXTCHECK(GL_EXT_polygon_offset);
    EXTCHECK(GL_EXT_texture);
    EXTCHECK(GL_EXT_guaranteed_to_fail);
    EXTCHECK(GLX_SGI_swap_control);
    EXTCHECK(GLX_SGI_video_sync);
    EXTCHECK(GLX_SGIS_multi_sample);

    REPORTEND;
    break;

  case 703:
    dumpIds();
    break;

/* Mess around with menus */

  case 800:
    if (glutGetMenu() != menu8)  /* just a test  */
      printf("glutGetMenu() returned unexpected value\n");
    glutAddMenuEntry("help", 101);
    glutAddMenuEntry("help", 101);
    glutAddMenuEntry("help", 101);
    glutAddMenuEntry("help", 101);
    glutAddMenuEntry("help", 101);
    break;

  case 801:
    glutAddSubMenu("shapes", menu3);
    glutAddSubMenu("shapes", menu3);
    glutAddSubMenu("shapes (a long string to break menus with)",
menu3);
    glutAddSubMenu("shapes", menu3);
    glutAddSubMenu("shapes", menu3);
    break;

  case 802:
    items = glutGet(GLUT_MENU_NUM_ITEMS);
    for (m = initItems + 1; m <= items; m++) {
      glutChangeToMenuEntry(m, "help", 101);
    }
    break;

  case 803:
    items = glutGet(GLUT_MENU_NUM_ITEMS);
    for (m = initItems + 1; m <= items; m++) {
      glutChangeToSubMenu(m, "shapes", menu3);
    }
```

```
    break;

  case 804:
    items = glutGet(GLUT_MENU_NUM_ITEMS);
    /* reverse order so renumbering not aproblem  */
    for (m = items; m >= initItems + 1; m--) {
      glutRemoveMenuItem(m);
    }
    break;

  case 805:
    menuButton[0] = !menuButton[0];
    attachMenus();
    break;

  case 806:
    menuButton[1] = !menuButton[1];
    attachMenus();
    break;

  case 807:
    menuButton[2] = !menuButton[2];
    attachMenus();
    break;

/* Direct menu items.  */

  case 100:
    exit(0);
    break;

  case 101:
    if (winId[5] == 0)
      makeWindow(5);
    else
      killWindow(5);
    break;

  case 9999:
    break;

  default:
    fprintf(stderr, "\007Unhandled case %d in menu callback\n",
value);
  }

}

/* redefineShapes - Remake the shapes display lists */

void
redefineShapes(int shape)
{
  int i;

#define C3               \
        switch(i)        \
        {                \
            case 0:      \
            case 3:      \
              C1;        \
```

```
              break;     \
                         \
            case 1:      \
            case 2:      \
            case 4:      \
            case 6:      \
            case 7:      \
              C2;        \
              break;     \
        }                \
        currentShape = shape

  for (i = 0; i < MAXWIN; i++) {
    if (winId[i]) {
      glutSetWindow(winId[i]);
      if (glIsList(i + 1))
        glDeleteLists(i + 1, 1);
      glNewList(i + 1, GL_COMPILE);

      switch (shape) {

#undef  C1
#define C1  glutSolidSphere(1.5, 10, 10)
#undef  C2
#define C2  glutWireSphere(1.5, 10, 10)

      case 0:
        C3;
        break;

#undef  C1
#define C1 glutSolidCube(2)
#undef  C2
#define C2 glutWireCube(2)

      case 1:
        C3;
        break;

#undef  C1
#define C1 glutSolidCone(1.5, 1.75, 10, 10);
#undef  C2
#define C2 glutWireCone(1.5, 1.75, 10, 10);

      case 2:
        C3;
        break;

#undef  C1
#define C1 glutSolidTorus(0.5, 1.1, 10, 10)
#undef  C2
#define C2 glutWireTorus(0.5, 1.1, 10, 10)

      case 3:
        C3;
        break;

#undef  C1
#define C1 glScalef(.8, .8, .8);glutSolidDodecahedron()
#undef  C2
#define C2 glScalef(.8, .8, .8);glutWireDodecahedron()
```

```
      case 4:
        C3;
        break;

#undef  C1
#define C1 glScalef(1.5, 1.5, 1.5);glutSolidOctahedron()
#undef  C2
#define C2 glScalef(1.5, 1.5, 1.5);glutWireOctahedron()

      case 5:
        C3;
        break;

#undef  C1
#define C1 glScalef(1.8, 1.8, 1.8);glutSolidTetrahedron()
#undef  C2
#define C2 glScalef(1.8, 1.8, 1.8);glutWireTetrahedron()

      case 6:
        C3;
        break;

#undef  C1
#define C1 glScalef(1.5, 1.5, 1.5);glutSolidIcosahedron()
#undef  C2
#define C2 glScalef(1.5, 1.5, 1.5);glutWireIcosahedron()

      case 7:
        C3;
        break;

#undef  C1
#define C1 glutSolidTeapot(1.5);
#undef  C2
#define C2 glutWireTeapot(1.5);

      case 8:
        C3;
        break;
      }
      glEndList();
    }
  }
}

/* positionWindow - Shift a window */

void
positionWindow(int index)
{
  int x, y;

  if (winId[index] == 0)
    return;

  glutSetWindow(winId[index]);
  x = glutGet(GLUT_WINDOW_X);
  y = glutGet(GLUT_WINDOW_Y);
  glutPositionWindow(x + 50, y + 50);
}
```

```
/* reshapeWindow - Change window size a little */

void
reshapeWindow(int index)
{
  int x, y;

  if (winId[index] == 0)
    return;
  glutSetWindow(winId[index]);
  x = glutGet(GLUT_WINDOW_WIDTH);
  y = glutGet(GLUT_WINDOW_HEIGHT);
/* glutReshapeWindow(x * (index % 2? 0.8: 1.2), y * (index % 2?

   1.2: 0.8)); */
  glutReshapeWindow((int) (x * 1.0), (int) (y * 0.8));
}

/* iconifyWindow - Iconify a window */

void
iconifyWindow(int index)
{
  if (winId[index] == 0)
    return;
  glutSetWindow(winId[index]);
  glutIconifyWindow();
}

/* showWindow - Show a window (map or uniconify it) */

void
showWindow(int index)
{
  if (winId[index] == 0)
    return;
  glutSetWindow(winId[index]);
  glutShowWindow();
}

/* hideWindow - Hide a window (unmap it) */

void
hideWindow(int index)
{
  if (winId[index] == 0)
    return;
  glutSetWindow(winId[index]);
  glutHideWindow();
}

/* pushWindow - Push a window */

void
pushWindow(int index)
{
  if (winId[index] == 0)
    return;
  glutSetWindow(winId[index]);
  glutPushWindow();
```

```
}

/* popWindow - Pop a window */

void
popWindow(int index)
{
  if (winId[index] == 0)
    return;
  glutSetWindow(winId[index]);
  glutPopWindow();
}

/* drawScene - Draw callback, triggered by expose events etc.
   in GLUT. */

void
drawScene(void)
{
  int winIndex;

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  winIndex = idToIndex(glutGetWindow());
  /* printf("drawScene for index %d, id %d\n", winIndex,
     glutGetWindow());  */

  glPushMatrix();
  glLineWidth(lineWidth);
  if (backdrop)
    glCallList(100);

  /* Left button spinning  */

  trackBall(APPLY, 0, 0, 0, 0);

  /* Apply continuous spinning  */

  glRotatef(angle, 0, 1, 0);

  glCallList(winIndex + 1);
  glPopMatrix();

  if (text[winIndex])
    showText();

  glutSwapBuffers();
}

/* showText - Render some text in the current GLUT window */

void
showText(void)
{
  glMatrixMode(GL_PROJECTION);
  glPushMatrix();
  glLoadIdentity();
  gluOrtho2D(0, 100, 0, 100);
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glLoadIdentity();
```

```
  glColor3f(1.0, 1.0, 1.0);
  glIndexi(7);

  glDisable(GL_DEPTH_TEST);
  glDisable(GL_LIGHTING);

  glLineWidth(lineWidth);

  textString(1, 1, "GLUT_BITMAP_8_BY_13", GLUT_BITMAP_8_BY_13);
  textString(1, 5, "GLUT_BITMAP_9_BY_15", GLUT_BITMAP_9_BY_15);
  textString(1, 10, "GLUT_BITMAP_TIMES_ROMAN_10",
GLUT_BITMAP_TIMES_ROMAN_10);
  textString(1, 15, "GLUT_BITMAP_TIMES_ROMAN_24",
GLUT_BITMAP_TIMES_ROMAN_24);

  strokeString(1, 25, "GLUT_STROKE_ROMAN", GLUT_STROKE_ROMAN);
  strokeString(1, 35, "GLUT_STROKE_MONO_ROMAN",
GLUT_STROKE_MONO_ROMAN);

  glEnable(GL_DEPTH_TEST);
  glEnable(GL_LIGHTING);

  glMatrixMode(GL_PROJECTION);
  glPopMatrix();
  glMatrixMode(GL_MODELVIEW);
  glPopMatrix();

}

/* textString - Bitmap font string */

void
textString(int x, int y, char *msg, void *font)
{
  glRasterPos2f(x, y);
  while (*msg) {
    glutBitmapCharacter(font, *msg);
    msg++;
  }
}

/* strokeString - Stroke font string */

void
strokeString(int x, int y, char *msg, void *font)
{
  glPushMatrix();
  glTranslatef(x, y, 0);
  glScalef(.04, .04, .04);
  while (*msg) {
    glutStrokeCharacter(font, *msg);
    msg++;
  }
  glPopMatrix();
}

/* idleFunc - GLUT idle func callback - animates windows */

void
idleFunc(void)
```

```
{
  int i;

  if (!leftDown && !middleDown)
    angle += 1;
    angle = angle % 360;

  for (i = 0; i < MAXWIN; i++) {
    if (winId[i] && winVis[i] && !winFreeze[i]) {
      glutSetWindow(winId[i]);
      glutPostRedisplay();
    }
  }
}

/* reshapeFunc - Reshape callback. */

void
reshapeFunc(int width, int height)
{
  int winId;
  float aspect;

  winId = glutGetWindow();
  PR("reshape callback for window id %d \n", winId);

  glViewport(0, 0, width, height);
  aspect = (float) width / height;

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(40.0, aspect, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
}

/* visible - Visibility callback. Turn off rendering in
   invisible windows */

void
visible(int state)
{
  int winId;
  static GLboolean someVisible = GL_TRUE;

  winId = glutGetWindow();
  /* printf("visible: state = %d \n", state);  */

  if (state == GLUT_VISIBLE) {
    PR("Window id %d visible \n", winId);
    winVis[idToIndex(winId)] = GL_TRUE;
  } else {
    PR("Window %d not visible \n", winId);
    winVis[idToIndex(winId)] = GL_FALSE;
  }

  if ((winVis[0] == GL_FALSE) && (winVis[1] == GL_FALSE) &&
(winVis[2] == GL_FALSE)
    && (winVis[3] == GL_FALSE) && (winVis[6] == GL_FALSE) &&
(winVis[7] ==
      GL_FALSE)) {
    glutIdleFunc(NULL);
```

```
    PR("All windows not visible; idle func disabled\n");
    someVisible = GL_FALSE;
  } else {
    if (!someVisible) {
      PR("Some windows now visible; idle func enabled\n");
      someVisible = GL_TRUE;
      if (animation)
        glutIdleFunc(idleFunc);
    }
  }
}

/* keyFunc - Ascii key callback */

/* ARGSUSED1 */
void
keyFunc(unsigned char key, int x, int y)
{
  int i, ii;

  if (debug || showKeys)
    printf("Ascii key '%c' 0x%02x\n", key, key);

  switch (key) {
  case 0x1b:
    exit(0);
    break;

  case 'a':
    demoMode = !demoMode;
    if (demoMode)
      autoDemo(-2);
    break;

  case 's':
    AUTODELAY = AUTODELAY * 0.666;
    break;

  case 'S':
    AUTODELAY = AUTODELAY * 1.5;
    break;

  case 'q':
    killWindow(idToIndex(glutGetWindow()));
    break;

  case 'k':
    showKeys = !showKeys;
    break;

  case 'p':
    demoMode = !demoMode;
    if (demoMode)
      autoDemo(-999);
    break;

  case 'D':
    debug = !debug;
    break;

  case 'd':
```

```
    dumpIds();
    break;

  case 'h':
    if (winId[5] == 0)
      makeWindow(5);
    else
      killWindow(5);
    break;

  case 't':
    ii = idToIndex(glutGetWindow());
    text[ii] = !text[ii];
    break;

  case 'r':
    trackBall(RESET, 0, 0, 0, 0);
    break;

  case 'l':
    lineWidth += 1;
    updateAll();
    break;

  case 'L':
    lineWidth -= 1;
    if (lineWidth < 1)
      lineWidth = 1;
    updateAll();
    break;

  case '0':
  case '1':
  case '2':
  case '3':
  case '4':
  case '6':
    i = key - '0';
    winVis[i] = !winVis[i];
    break;

  case ')':
    makeWindow(0);
    break;

  case '!':
    makeWindow(1);
    break;

  case '@':
    makeWindow(2);
    break;

  case '#':
    makeWindow(3);
    break;

  }
}

/* specialFunc - Special keys callback (F keys, cursor keys
```

```
   etc. */

/* ARGSUSED1 */
void
specialFunc(int key, int x, int y)
{
  if (debug || showKeys)
    printf("Special key %d\n", key);

  switch (key) {
  case GLUT_KEY_PAGE_DOWN:
    scrollLine += 10;
    updateHelp();
    updateText();
    break;

  case GLUT_KEY_PAGE_UP:
    scrollLine -= 10;
    updateHelp();
    updateText();
    break;

  case GLUT_KEY_DOWN:
    scrollLine += 1;
    updateHelp();
    updateText();
    break;

  case GLUT_KEY_UP:
    scrollLine -= 1;
    updateHelp();
    updateText();
    break;

  case GLUT_KEY_HOME:
    scrollLine = 0;
    updateHelp();
    updateText();
    break;

  case GLUT_KEY_END:
    scrollLine = 9999;
    updateHelp();
    updateText();
    break;

  case GLUT_KEY_RIGHT:
    scrollCol -= 1;
    updateHelp();
    updateText();
    break;

  case GLUT_KEY_LEFT:
    scrollCol += 1;
    updateHelp();
    updateText();
    break;
  }
}

/* mouseFunc - Mouse button callback */
```

```
void
mouseFunc(int button, int state, int x, int y)
{
  PR("Mouse button %d, state %d, at pos %d, %d\n", button, state, x,
y);

  trackBall(MOUSEBUTTON, button, state, x, y);
}

/* motionFunc - Mouse movement (with a button down) callback */

void
motionFunc(int x, int y)
{
  PR("Mouse motion at %d, %d\n", x, y);

  trackBall(MOUSEMOTION, 0, 0, x, y);

  glutPostRedisplay();
}

/* passiveMotionFunc - Mouse movement (with no button down)
   callback */

void
passiveMotionFunc(int x, int y)
{
  printf("Mouse motion at %d, %d\n", x, y);
}

/* entryFunc - Window entry event callback */

void
entryFunc(int state)
{
  int winId = glutGetWindow();
  PR("Entry event: window id %d (index %d), state %d \n", winId,
idToIndex(
      winId), state);
}

/* menuStateFunc - Callback to tell us when menus are popped
   up/down. */

int menu_state = GLUT_MENU_NOT_IN_USE;

void
menuStateFunc(int state)
{
  printf("menu stated = %d\n", state);
  menu_state = state;

  if (glutGetWindow() == 0) {
    PR("menuStateFunc: window invalid\n");
    return;
  }
  PR("Menus are%sin use\n", state == GLUT_MENU_IN_USE ? " " : " not
");

  if ((state == GLUT_MENU_IN_USE) && menuFreeze)
```

```
    glutIdleFunc(NULL);
  else if (animation)
    glutIdleFunc(idleFunc);
}

/* timerFunc - General test of global timer */

void
timerFunc(int value)
{
  printf("timer callback: value %d\n", value);
  if (timerOn) {
    glutTimerFunc(DELAY, timerFunc, 1);
  }
}

#if 0
/* delayedReinstateMenuStateCallback - Hack to reinstate
   MenuStateCallback after a while.  */

void
delayedReinstateMenuStateCallback(int state)
{
  glutMenuStateFunc(menuStateFunc);
}

#endif

/* setInitDisplayMode - update display modes from display mode
   menu */

void
setInitDisplayMode(void)
{
  int i;

  displayMode = 0;

  for (i = 0; i < MODES; i++) {
    if (modes[i]) {
      /* printf("Requesting %s \n", modeNames[i]);  */
      displayMode |= glutMode[i];
    }
  }

  glutInitDisplayMode(displayMode);

  createMenu6();
  if (!glutGet(GLUT_DISPLAY_MODE_POSSIBLE))
    warning("This display mode not supported\n");
}

/* makeWindow - Create one of the windows */

void
makeWindow(int index)
{
  char str[99];

  if (winId[index] != 0) {
    /* warning("Attempt to create window which is already
```

```
      created");  */
    return;
  }
  switch (index) {

  case 0:              /* ordinary RGB windows  */
  case 1:
  case 2:
  case 3:

    setInitDisplayMode();
    glutInitWindowPosition(pos[index][0], pos[index][1]);
    glutInitWindowSize(size[index][0], size[index][1]);
    winId[index] = glutCreateWindow(" ");
    PR("Window %d id = %d \n", index, winId[index]);
    gfxInit(index);

    addCallbacks();

    sprintf(str, "window %d (RGB)", index);
    glutSetWindowTitle(str);
    sprintf(str, "icon %d", index);
    glutSetIconTitle(str);
    glutSetMenu(menu1);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    break;

  case 4:              /* subwindow  */

    setInitDisplayMode();
    winId[index] = glutCreateSubWindow(winId[0], pos[index][0],
pos[index]
      [1], size[index][0], size[index][1]);
    PR("Window %d id = %d \n", index, winId[index]);
    gfxInit(index);
    glutDisplayFunc(drawScene);
    glutVisibilityFunc(visible);
    glutReshapeFunc(reshapeFunc);

    break;

  case 5:              /* help window  */
  case 8:              /* text window  */
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition(pos[index][0], pos[index][1]);
    glutInitWindowSize(size[index][0], size[index][1]);
    winId[index] = glutCreateWindow(" ");
    PR("Window %d id = %d \n", index, winId[index]);

    /* addCallbacks();  */
    glutKeyboardFunc(keyFunc);
    glutSpecialFunc(specialFunc);

    glClearColor(0.15, 0.15, 0.15, 1.0);
    glColor3f(1, 1, 1);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 300, 0, 100);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
```

```
    if (index == 5) {
      glutDisplayFunc(updateHelp);
      glutSetWindowTitle("help (RGB) win 5");
      glutSetIconTitle("help");
    } else {
      glutDisplayFunc(updateText);
      glutSetWindowTitle("text (RGB) win 8");
      glutSetIconTitle("text");
    }
    glutSetMenu(menu1);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    break;

  case 6:              /* color index window  */
  case 7:              /* color index window  */
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_INDEX | GLUT_DEPTH);
         if (glutGet(GLUT_DISPLAY_MODE_POSSIBLE)) {
                glutInitWindowPosition(pos[index][0],
pos[index][1]);
                glutInitWindowSize(size[index][0], size[index][1]);
                winId[index] = glutCreateWindow(" ");
                PR("Window %d id = %d \n", index, winId[index]);

                gfxInit(index);

                addCallbacks();

                sprintf(str, "window %d (color index)", index);
                glutSetWindowTitle(str);
                sprintf(str, "icon %d", index);
                glutSetIconTitle(str);
                glutSetMenu(menu1);
                glutAttachMenu(GLUT_RIGHT_BUTTON);
         }
    break;

  }
}

/* killWindow - Kill one of the main windows */

void
killWindow(int index)
{
  int i;

  if (winId[index] == 0) {
    /* fprintf(stderr, "Attempt to kill invalid window in
       killWindow\n");  */
    return;
  }
  PR("Killing win %d\n", index);
  glutSetWindow(winId[index]);

  /* Disable all callbacks for safety, although
     glutDestroyWindow  should do this.  */

  removeCallbacks();

  glutDestroyWindow(winId[index]);
  winId[index] = 0;
```

```
  winVis[index] = GL_FALSE;

#if 0
  /* If we reinstate the menu state func here, prog breaks.  So
     reinstate it a little later.  */
  glutTimerFunc(MENUDELAY, delayedReinstateMenuStateCallback, 1);
#endif

  if (index == 5) {     /* help  */
    scrollLine = 0;
    scrollCol = 0;
  }
  if (index == 8) {     /* text window  */
    for (i = 0; textPtr[i] != NULL; i++) {
      free(textPtr[i]); /* free the text strings  */
      textPtr[i] = NULL;
    }
  }
}

/* addCallbacks - Add some standard callbacks after creating a
   window */

void
addCallbacks(void)
{
  glutDisplayFunc(drawScene);
  glutVisibilityFunc(visible);
  glutReshapeFunc(reshapeFunc);
  glutKeyboardFunc(keyFunc);
  glutSpecialFunc(specialFunc);
  glutMouseFunc(mouseFunc);
  glutMotionFunc(motionFunc);
  glutEntryFunc(entryFunc);

/* Callbacks for exotic input devices. Must get my dials &
   buttons back. */

  glutSpaceballMotionFunc(spaceballMotionCB);
  glutSpaceballRotateFunc(spaceballRotateCB);
  glutSpaceballButtonFunc(spaceballButtonCB);

  glutButtonBoxFunc(buttonBoxCB);
  glutDialsFunc(dialsCB);

  glutTabletMotionFunc(tabletMotionCB);
  glutTabletButtonFunc(tabletButtonCB);
}

/* removeCallbacks - Remove all callbacks before destroying a
   window. GLUT probably does this  anyway but we'll be safe. */

void
removeCallbacks(void)
{
  glutVisibilityFunc(NULL);
  glutReshapeFunc(NULL);
  glutKeyboardFunc(NULL);
  glutSpecialFunc(NULL);
  glutMouseFunc(NULL);
  glutMotionFunc(NULL);
```

```
  glutEntryFunc(NULL);
}

/* updateHelp - Update the help window after user scrolls. */

void
updateHelp(void)
{
  static char *helpPtr[] =
  {
    "(Use PGUP, PGDN, HOME, END, arrows to scroll help text)
",
    "
",
    "A demo program for GLUT.
",
    "G Edwards, Aug 95
",
    "Exercises 99% of GLUT calls
",
    VERSIONLONG,
    "
",
    "This text uses GLUT_STROKE_MONO_ROMAN font, a built-in vector
font.",
    "(Try resizing the help window).
",
    "
",
    "Keys:
",
    " esc   quit
",
    " t     toggle text on/off in each window
",
    " h     toggle help
",
    " q     quit current window
",
    " a     auto demo
",
    " p     pause/unpause demo
",
    " l     increase line width (gfx & stroke text)
",
    " L     decrease line width (gfx & stroke text)
",
    " r     reset transforms
",
    " k     show keyboard events
",
    " D     show all events
",
    "
",
    "Mouse:
",
    " Left button:    rotate
",
    " Middle button:  pan
",
```

```
    " Left + middle:  zoom
",
    NULL};

  updateScrollWindow(5, helpPtr);
}

/* updateText - Update a text window */

void
updateText(void)
{
  int i;

  if (textPtr[0] == NULL) {
    for (i = 0; i < 20; i++) {
      textPtr[i] = (char *) malloc(50);
      strcpy(textPtr[i], "no current text");
    }
    textPtr[20] = NULL;
  }
  updateScrollWindow(8, textPtr);
}

/* updateScrollWindow */

void
updateScrollWindow(int index, char **ptr)
{
  int i, j, lines = 0;

  if (winId[index] == 0)
    return;

  glutSetWindow(winId[index]);

  for (i = 0; ptr[i] != NULL; i++)
    lines++;

  if (scrollLine < 0)
    scrollLine = 0;
  if (scrollLine > (lines - 5))
    scrollLine = lines - 5;

  glClear(GL_COLOR_BUFFER_BIT);

  glLineWidth(lineWidth);

  for (i = scrollLine, j = 1; ptr[i] != NULL; i++, j++)
    strokeString(scrollCol * 50, 100 - j * 6, ptr[i],
      GLUT_STROKE_MONO_ROMAN);

  glutSwapBuffers();

}

/* updateAll - Update all visible windows after soem global
   change, eg. line width */

void
updateAll(void)
```

```
{
  int i;

  if (winId[5] != 0)
    updateHelp();

  if (winId[8] != 0)
    updateText();

  for (i = 0; i < MAXWIN; i++)
    if (winId[i]) {
      glutSetWindow(winId[i]);
      glutPostRedisplay();
    }
}

/* idToIndex - Convert GLUT window id to our internal index */

int
idToIndex(int id)
{
  int i;
  for (i = 0; i < MAXWIN; i++) {
    if (winId[i] == id)
      return i;
  }
  fprintf(stderr, "error: id %d not found \n", id);
  return (-1);
}

/* warning - warning messages */

void
warning(char *msg)
{
  fprintf(stderr, "\007");

  if (debug) {
    fprintf(stderr, "%s", msg);
    if (msg[strlen(msg)] != '\n')
      fprintf(stderr, "%s", "\n");
  }
}

/* dumpIds - Debug: dump some internal data  */

void
dumpIds(void)
{
  int i, j;

  printf("\nInternal data:\n");

  for (i = 0; i < MAXWIN; i++)
    printf("Index %d, glut win id %d, visibility %d\n", i, winId[i],
      winVis[i]);

  for (i = 0; i < MAXWIN; i++) {
    if (winId[i])
      glutSetWindow(winId[i]);
    else {
```

```
      printf("index %d - no glut window\n", i);
      continue;
    }

    for (j = 1; j <= MAXWIN; j++)
      printf("Index %d, display list %d %s defined\n", i, j,
glIsList(j) ?
        "is " : "not");
  }
}

/* autoDemo - Run auto demo/test This is a bit tricky. We need
   to start a timer sequence which progressively orders things
   to be done. The work really gets done when we return from
   our callback. Have to think about the event loop / callback
   design here. */

void
autoDemo(int value)
{

#define STEP(a, b)  \
    case a:         \
        action(a);  \
        glutTimerFunc(AUTODELAY * b, autoDemo, next(a); \
        break;

  static int index = 0;
  static int count = 0;
  static int restartValue = -2;

  if (value == -999)
    value = restartValue;

  restartValue = value;

#define AUTODELAY2 (unsigned int) (AUTODELAY*0.66)

  /* fprintf(stderr, "autoDemo: value %d \n", value);  */

  if (!demoMode)
    return;

  if (menu_state == GLUT_MENU_IN_USE) {
    glutTimerFunc(AUTODELAY / 2, autoDemo, value);
    return;
  }
  switch (value) {

/* Entry point; kill off existing windows. */

  case -2:
    killAllWindows();
    glutTimerFunc(AUTODELAY / 2, autoDemo, 1);
    break;

/* Start making windows */

  case -1:
    makeWindow(0);
    glutTimerFunc(AUTODELAY, autoDemo, 0);  /* skip case 0
```

```
                                                    first time  */
    break;

/* Change shape & backdrop */

  case 0:
    currentShape = (currentShape + 1) % 9;
    redefineShapes(currentShape);
    count += 1;
    if (count % 2)
      backdrop = !backdrop;
    glutTimerFunc(AUTODELAY, autoDemo, 1);
    break;

/* Keep making windows */

  case 1:
    makeWindow(1);
    glutTimerFunc(AUTODELAY, autoDemo, 2);
    break;

  case 2:
    makeWindow(2);
    glutTimerFunc(AUTODELAY, autoDemo, 3);
    break;

  case 3:
    makeWindow(3);
    glutTimerFunc(AUTODELAY, autoDemo, 4);
    break;

  case 4:
    makeWindow(4);
    glutTimerFunc(AUTODELAY, autoDemo, 5);
    break;

  case 5:
    makeWindow(5);
    glutTimerFunc(AUTODELAY * 2, autoDemo, 51);
    break;

  case 51:
    makeWindow(6);
    glutTimerFunc(AUTODELAY * 2, autoDemo, 52);
    break;

  case 52:
    makeWindow(7);
    glutTimerFunc(AUTODELAY * 2, autoDemo, 53);
    break;

/* Kill last 3 windows, leave 4 up. */

  case 53:
    killWindow(7);
    glutTimerFunc(AUTODELAY, autoDemo, 54);
    break;

  case 54:
    killWindow(6);
    glutTimerFunc(AUTODELAY, autoDemo, 6);
```

```
    break;

  case 6:
    killWindow(5);
    glutTimerFunc(AUTODELAY, autoDemo, 7);
    break;

  case 7:
    killWindow(4);
    glutTimerFunc(AUTODELAY, autoDemo, 700);
    break;

/* Change shape again */

  case 700:
    currentShape = (currentShape + 1) % 9;
    redefineShapes(currentShape);
    glutTimerFunc(AUTODELAY, autoDemo, 701);
    break;

/* Cycle 4 main windows through various window ops.  */

  case 701:
    positionWindow(index);
    index = (index + 1) % 4;
    glutTimerFunc(AUTODELAY2, autoDemo, index > 0 ? 701 : 702);
    break;

  case 702:
    reshapeWindow(index);
    index = (index + 1) % 4;
    glutTimerFunc(AUTODELAY2, autoDemo, index > 0 ? 702 : 703);
    break;

  case 703:
    iconifyWindow(index);
    index = (index + 1) % 4;
    glutTimerFunc(AUTODELAY2, autoDemo, index > 0 ? 703 : 704);
    break;

  case 704:
    showWindow(index);
    index = (index + 1) % 4;
    glutTimerFunc(AUTODELAY2, autoDemo, index > 0 ? 704 : 705);
    break;

  case 705:
    hideWindow(index);
    index = (index + 1) % 4;
    glutTimerFunc(AUTODELAY2, autoDemo, index > 0 ? 705 : 706);
    break;

  case 706:
    showWindow(index);
    index = (index + 1) % 4;
    glutTimerFunc(AUTODELAY2, autoDemo, index > 0 ? 706 : 707);
    break;

  case 707:
    pushWindow(index);
    index = (index + 1) % 4;
```

```
    glutTimerFunc(AUTODELAY2, autoDemo, index > 0 ? 707 : 708);
    break;

  case 708:
    popWindow(index);
    index = (index + 1) % 4;
    glutTimerFunc(AUTODELAY2, autoDemo, index > 0 ? 708 : 8);
    break;

/* Kill all windows */

  case 8:
    killWindow(3);
    glutTimerFunc(AUTODELAY, autoDemo, 9);
    break;

  case 9:
    killWindow(2);
    glutTimerFunc(AUTODELAY, autoDemo, 10);
    break;

  case 10:
    killWindow(1);
    glutTimerFunc(AUTODELAY, autoDemo, 11);
    break;

  case 11:
    killWindow(0);
    glutTimerFunc(AUTODELAY, autoDemo, -1);  /* back to start  */
    break;
  }

}

/* attachMenus - Attach/detach menus to/from mouse buttons */

void
attachMenus(void)
{
  int i, b;
  int button[3] =
  {GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON};

  for (i = 0; i < MAXWIN; i++) {
    if (winId[i] != 0) {
      for (b = 0; b < 3; b++) {
        glutSetWindow(winId[i]);
        glutSetMenu(menu1);
        if (menuButton[b])
          glutAttachMenu(button[b]);
        else
          glutDetachMenu(button[b]);
      }
    }
  }
}

/* killAllWindows - Kill all windows (except 0) */

void
killAllWindows(void)
```

```
{
  int w;

  for (w = 1; w < MAXWIN; w++)
    if (winId[w])
      killWindow(w);
}

/* makeAllWindows - Make all windows */

void
makeAllWindows(void)
{
  int w;

  for (w = 0; w < MAXWIN; w++)
    if (!winId[w])
      makeWindow(w);
}

/* checkArgs - Check command line args */

void
checkArgs(int argc, char *argv[])
{
  int argp;
  GLboolean quit = GL_FALSE;
  GLboolean error = GL_FALSE;

#define AA argv[argp]

#if 0
#define NEXT argp++;      \
            if(argp >= argc) \
            {                \
               Usage();      \
               Exit(1);      \
            }
#endif

  argp = 1;
  while (argp < argc) {
    if (match(AA, "-help")) {
      commandLineHelp();
      quit = GL_TRUE;
    } else if (match(AA, "-version")) {
      printf(VERSIONLONG "\n");
      quit = GL_TRUE;
    } else if (match(AA, "-auto")) {
      demoMode = GL_TRUE;
    } else if (match(AA, "-scale")) {
      argp++;
      scaleFactor = atof(argv[argp]);
    } else {
      fprintf(stderr, "Unknown arg: %s\n", AA);
      error = GL_TRUE;
      quit = GL_TRUE;
    }
    argp++;
  }
```

```
  if (error) {
    commandLineHelp();
    exit(1);
  }
  if (quit)
    exit(0);
}

/* commandLineHelp - Command line help */

void
commandLineHelp(void)
{
  printf("Usage:\n");
  printf(" -h[elp]            this stuff\n");
  printf(" -v[ersion]         show version\n");
  printf(" -a[uto]            start in auto demo mode\n");
  printf(" -s[cale] f         scale windows by f\n");
  printf("Standard GLUT args:\n");
  printf(" -iconic            start iconic\n");
  printf(" -display DISP      use display DISP\n");
  printf(" -direct            use direct rendering (default)\n");
  printf(" -indirect          use indirect rendering\n");
  printf(" -sync              use synchronous X protocol\n");
  printf(" -gldebug           check OpenGL errors\n");
  printf(" -geometry WxH+X+Y  standard X window spec (overridden
here) \n");
}

/* match - Match a string (any unique substring). */

GLboolean
match(char *arg, char *t)
{
  if (strstr(t, arg))
    return GL_TRUE;
  else
    return GL_FALSE;
}

/* scaleWindows - Scale initial window sizes ansd positions */

void
scaleWindows(float scale)
{
  int i;

  for (i = 0; i < MAXWIN; i++) {
    pos[i][0] = pos[i][0] * scale;
    pos[i][1] = pos[i][1] * scale;
    size[i][0] = size[i][0] * scale;
    size[i][1] = size[i][1] * scale;
  }
}

/* trackBall - A simple trackball (not with proper rotations). */

/** A simple trackball with spin = left button
                           pan  = middle button
                           zoom = left + middle
```

```
   Doesn't have proper trackball rotation, ie axes which remain fixed
in
   the scene. We should use the trackball code from 4Dgifts. */

#define STARTROTATE(x, y)     \
{                             \
    startMX = x;              \
    startMY = y;              \
}

#define STOPROTATE(x, y)      \
{                             \
    steadyXangle = varXangle; \
    steadyYangle = varYangle; \
}

#define STARTPAN(x, y)        \
{                             \
    startMX = x;              \
    startMY = y;              \
}

#define STOPPAN(x, y)         \
{                             \
    steadyX = varX;           \
    steadyY = varY;           \
}

#define STARTZOOM(x, y)       \
{                             \
    startMX = x;              \
    startMY = y;              \
}

#define STOPZOOM(x, y)        \
{                             \
    steadyZ = varZ;           \
}

static float
fixAngle(float angle)
{
  return angle - floor(angle / 360.0) * 360.0;
}

void
trackBall(int mode, int button, int state, int x, int y)
{
  static int startMX = 0, startMY = 0;  /* initial mouse pos  */
  static int deltaMX = 0, deltaMY = 0;  /* initial mouse pos  */
  static float steadyXangle = 0.0, steadyYangle = 0.0;
  static float varXangle = 0.0, varYangle = 0.0;
  static float steadyX = 0.0, steadyY = 0.0, steadyZ = 0.0;
  static float varX = 0.0, varY = 0.0, varZ = 0.0;

  switch (mode) {

  case RESET:
    steadyXangle = steadyYangle = steadyX = steadyY = steadyZ = 0.0;
    break;
```

```
  case MOUSEBUTTON:

    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN &&
!middleDown) {
      STARTROTATE(x, y);
      leftDown = GL_TRUE;
    } else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN &&
      middleDown) {
      STOPPAN(x, y);
      STARTZOOM(x, y);
      leftDown = GL_TRUE;
    } else if (button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN &&
      !leftDown) {
      STARTPAN(x, y);
      middleDown = GL_TRUE;
    } else if (button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN &&
      leftDown) {
      STOPROTATE(x, y);
      STARTZOOM(x, y);
      middleDown = GL_TRUE;
    } else if (state == GLUT_UP && button == GLUT_LEFT_BUTTON &&
!middleDown) {
      STOPROTATE(x, y);
      leftDown = GL_FALSE;
    } else if (state == GLUT_UP && button == GLUT_LEFT_BUTTON &&
middleDown) {
      STOPZOOM(x, y);
      STARTROTATE(x, y);
      leftDown = GL_FALSE;
    } else if (state == GLUT_UP && button == GLUT_MIDDLE_BUTTON &&
!leftDown) {
      STOPPAN(x, y);
      middleDown = GL_FALSE;
    } else if (state == GLUT_UP && button == GLUT_MIDDLE_BUTTON &&
leftDown) {
      STOPZOOM(x, y);
      STARTROTATE(x, y);
      middleDown = GL_FALSE;
    }
    break;

  case APPLY:

    if (leftDown && !middleDown) {
      glTranslatef(steadyX, steadyY, steadyZ);
      glRotatef(varXangle, 0, 1, 0);
      glRotatef(varYangle, 1, 0, 0);
    }
    /* Middle button pan  */

    else if (middleDown && !leftDown) {
      glTranslatef(varX, varY, steadyZ);
      glRotatef(steadyXangle, 0, 1, 0);
      glRotatef(steadyYangle, 1, 0, 0);
    }
    /* Left + middle zoom.  */

    else if (leftDown && middleDown) {
      glTranslatef(steadyX, steadyY, varZ);
      glRotatef(steadyXangle, 0, 1, 0);
      glRotatef(steadyYangle, 1, 0, 0);
```

```
    }
    /* Nothing down.  */

    else {
      glTranslatef(steadyX, steadyY, steadyZ);
      glRotatef(steadyXangle, 0, 1, 0);
      glRotatef(steadyYangle, 1, 0, 0);
    }
    break;

  case MOUSEMOTION:

    deltaMX = x - startMX;
    deltaMY = startMY - y;

    if (leftDown && !middleDown) {
      varXangle = fixAngle(steadyXangle + deltaMX);
      varYangle = fixAngle(steadyYangle + deltaMY);
    } else if (middleDown && !leftDown) {
      varX = steadyX + deltaMX / 100.0;
      varY = steadyY + deltaMY / 100.0;
    } else if (leftDown && middleDown) {
      varZ = steadyZ - deltaMY / 50.0;
    }
    break;
  }

}

/* Callbacks for exotic input devices. These have not been
   tested yet owing to the usual complete absence of such
   devices in the UK support group. */

/* spaceballMotionCB */

void
spaceballMotionCB(int x, int y, int z)
{
  printf("spaceballMotionCB: translations are X %d, Y %d, Z %d\n", x,
y, z);
}

/* spaceballRotateCB */

void
spaceballRotateCB(int x, int y, int z)
{
  printf("spaceballRotateCB: rotations are X %d, Y %d, Z %d\n", x, y,
z);
}

/* spaceballButtonCB */

void
spaceballButtonCB(int button, int state)
{
  printf("spaceballButtonCB: button %d, state %d\n", button, state);
}

/* buttonBoxCB */
```

```
void
buttonBoxCB(int button, int state)
{
  printf("buttonBoxCB: button %d, state %d\n", button, state);
}

/* dialsCB */

void
dialsCB(int dial, int value)
{
  printf("dialsCB: dial %d, value %d\n", dial, value);
}

/* tabletMotionCB */

void
tabletMotionCB(int x, int y)
{
  printf("tabletMotionCB: X %d, Y %d\n", x, y);
}

/* tabletButtonCB */

/* ARGSUSED2 */
void
tabletButtonCB(int button, int state, int dummy1, int dummy2)
{
  printf("tabletButtonCB: button %d, state %d\n", button, state);
}
```

تعليمات مكتبة الـــ OpenGL:

توجد مجموعة من تعليمات الـــ OpenGL تقوم بنفس العمل إلا أن الاختلاف فيها يكون بنوع الوسطاء الممررة لها.

مثلاً :

```
void glVertex2{s}{v} (TYPE x, TYPE y);
```

يشير الرقم الموجود في نهاية التعليمة إلى عدد الوسطاء التي يمكن أن تأخذها التعليمة، ويشير الرمز الموجود بين قوسين إلى نوع المعطيات التي تأخذها وسطاء هذه التعليمة أما بالنسبة للرمز v يشير أن الوسيط الممرر للتعليمة عبارة عن شعاع أو مصفوفة .

تقبل الـــ OpenGL ثمانية أنماط لوسطائها (متحولات دخل التابع) والتي ستوضح في الجدول :

Suffix	**Data Type**	**Typical Corresponding C-Language Type**	**OpenGL Type Definition**
b	8-bit integer	signed char	Glbyte
s	16-bit integer	Short	Glshort
i	32-bit integer	int or long	GLint, Glsizei
f	32-bit floating-point	Float	GLfloat, Glclampf
d	64-bit floating-point	Double	GLdouble, Glclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, Glboolean
us	16-bit unsigned integer	unsigned short	Glushort
ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint, GLenum, Glbitfield

وسنعرض بعض التعليمات الأساسية في الـــOpenGL في الشكل التالي:

Primitives	**الأساسيات:** تفيد في إنشاء أبسط الأشكال الهندسية مثل المضلع،النقاط ومنه يمكن الإنطلاق لرسم باقي الأشكال والنماذج.
• void **glBegin** (GLenum *mode*); • void **glEnd** (void); • void **glVertex2**{sifd}{v}(TYPE *x*, TYPE *y*); • void **glVertex3**{sifd}{v}(TYPE x, TYPE y, TYPE z); • void **glVertex4**{sifd}{v}(TYPE x, TYPE y, TYPE z, TYPE w); • void **glRect**{sifd}(TYPE x1, TYPE y1, TYPE x2, TYPE y2); • void **glRect**{sifd}v(const TYPE **v1*, const TYPE **v2*);	
• void **glEdgeFlag**(GLboolean flag); • void **glEdgeFlagv**(const GLboolean *flag);	تصف طريقة معالجة حواف المضلع
Coordinate Transformation	**تحويل الإحداثيات:** تفيد في إجراء عمليات الانسحاب والدوران والتقييس لجملة الإحداثيات المستخدمة بواسطة مصفوفات التحويل.
• void **glRotate**{fd}(TYPE angle, TYPE x, TYPE y, TYPE z); • void **glTranslate**{fd}(TYPE x, TYPE y, TYPE z); • void **glScale**{fd}(TYPE x, TYPE y, TYPE z); • void **glMultMatrix**{fd}(const TYPE *m); • void **glFrustum** (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far); • void **glOrtho** (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);	
• void **glLoadMatrix**{fd}(const TYPE **m*); • void **glLoadIdentity** (void);	تهيئة مصفوفة التحويلات بالقيم الافتراضية
• void **glMatrixMode**(GLenum mode); • void **glPushMatrix**(void); • void **glPopMatrix** (void);	معالجة مكدس مصفوفة التحويلات
• void **glDepthRange**(GLclampd near, GLclampd far); • void **glViewport**(GLint x, GLint y, GLsizei width, GLsizei height);	تعين مواصفات بوابة الرؤية
Coloring and Lighting	**التلوين والإضاءة:** تفيد في إضافة المنابع الضوئية للمشهد وتلوين المواد وإضاءتها.
• void **glColor3**{bsifd ubusui}{v}(TYPE red, TYPE green, TYPE blue); • void **glColor4**{bsifd ubusui}{v}(TYPE red, TYPE green, TYPE blue, TYPE alpha); • void **glIndex**{sifd}{v}(TYPE index); • void **glNormal3**{bsifd}{v}(TYPE nx, TYPE ny, TYPE nz);	تحديد اللون المستخدم و نظام الألوان

• void **glLight**{if}{v}(GLenum light, GLenum pname, TYPE param); • void **glMaterial**{if}{v}(GLenum face, GLenum pname, TYPE param); • void **glLightModel**{if}{v}(Glenum pname, TYPE param);	تحديد خصائص مصدر الإضاءة، خصائص الأجسام، تحديد خصائص نموذج الإضاءة
• void **glShadeModel**(GLenum mode);	تحديد خصائص نموذج التظليل المستخدم
• void **glFrontFace**(GLenum dir);	تحديد الوجه الأمامي للمضلع
• void **glColorMaterial**(GLenum face, GLenum mode);	تغيير خصائص الأجسام بتحديد الوجه المضاء ونموذج الإضاءة الناتج عن ذلك الجسم
• void **glGetLight**{if}v(GLenum light, GLenum pname, TYPE *params); • void **glGetMaterial**{if}v(GLenum face, GLenum pname, TYPE params);	الحصول على قيم المصدر الضوئي، وخصائص الأجسام
Clipping	**القطع:** تفيد في اقتطاع بعض أجزاء النموذج المعروض.
• void **glClipPlane** (GLenum plane, const GLdouble *equation);	إجراء القطع
• void **glGetClipPlane** (GLenum plane, Gldouble *equation);	تعيد المعامل plane أي السطح الذي تم اقتطاعه
Rasterization	**تحويل إلى خطوط مسح:** تفيد في تحويل الرسوم الشعاعية (الصور الموصفة بعناصرها الرياضية كالنقاط والخطوط) إلى صور مكافئة مؤلفة من مجموعات بقع تخزن ثم تعالج كمجموعة من البتات
• void **glRasterPos2**{sifd}{v}(TYPE x, TYPE y); • void **glRasterPos3**{sifd}{v}(TYPE x, TYPE y, TYPE z); • void **glRasterPos4**{sifd}{v}(TYPE x, TYPE y, TYPE z, TYPE w);	تحديد موقع الماسح
• void **glBitmap** (GLsizei width, GLsizei height, GLfloat xorig, GLfloat yorig, GLfloat xmove, GLfloat ymove, const GLubyte *bitmap);	تحديد خصائص الصورة
• void **glPointSize**(GLfloat size); • void **glLineWidth**(GLfloat width);	تحديد أبعاد نقطة أو خط (حجم النقطة، عرض الخط)

تعطي الخط أو المضلع نموذج منقط	• void **glLineStipple**(GLint factor, GLushort pattern); • void **glPolygonStipple**(const GLubyte *mask); • void **glGetPolygonStipple**(Glubyte *mask);
تحديد طريقة المسح الذي ستنفذ على المضلع	• void **glCullFace**(GLenum mode); • void **glPolygonMode**(GLenum face, GLenum mode);
Pixel Operations	العمليات على البكسلات:
اختيار المصدر لقراءة البكسلات وطباعتها	• void **glReadBuffer** (GLenum mode);
قراءة وطباعة ونسخ البكسلات	• void **glReadPixels**(GLint x, GLint y, GLsizei width, GLsizei height, GLenum format, GLenum type, GLvoid *pixels); • void **glDrawPixels**(GLsizei width, GLsizei height, GLenum format, GLenum type, const Glvoid *pixels); • void **glCopyPixels**(GLint x, GLint y, GLsizei width, GLsizei height, GLenum type);
الاستفسار عن الطريقة التي استخدمت لترميز ومعالجة البكسلات	• void **glPixelStore**{if}(GLenum pname, TYPE param); • void **glPixelTransfer**{if}(Glenum pname, TYPE param); • void **glPixelMap**{f usui}v(Glenum map, Glint size, const TYPE *values); • void **glGetPixelMap**{f usui}v(GLenum map, TYPE *values);
التحكم بالبكسلات الممسوحة	• void **glPixelZoom** (GLfloat xfactor, GLfloat yfactor);
Texture Mapping	تلبيس نسيجي: تظليل أو إضافة صفات أخرى إلى سطح الصورة لإعطائها بعض الخدع البصرية.
التحكم بطريقة تطبيق الإكساء على أجزاء الصورة أجزاء = fragments	• void **glTexParameter**{if}{v}(Glenum target, GLenum pname, TYPE param); • void **glTexEnv**{if}{v} (GLenum target, GLenum pname, TYPE param);

• void **glTexCoord1**{sifd}{v}(TYPE s); • void **glTexCoord2**{sifd}{v}(TYPE s, TYPE t); • void **glTexCoord3**{sifd}{v}(TYPE s, TYPE t, TYPE r); • void **glTexCoord4**{sifd}{v}(TYPE s, TYPE t, TYPE r, TYPE q);	تحديد إحداثيات موقع الإكساء
• void **glTexGen**{ifd}{v}(Glenum coord, GLenum pname, TYPE param);	التحكم بطريقة توليد إحداثيات الإكساء
• void **glTexImage1D**(GLenum target, GLint level, Glint components, GLsizei width, GLint border, GLenum format, GLenum type, const GLvoid *pixels); • void **glTexImage2D**(GLenum target, GLint level, GLint components, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *pixels);	تحديد الخصائص المتعلقة بأبعاد الصورة الخاضعة للإكساء
Fog	الضباب:
void **glFog**{if}{v} (GLenum pname, TYPE param);	إضافة الضباب إلى المشهد وتحديد خصائصه
Frame Buffer Operations	العمليات على المخزن المؤقت:
• void **glScissor** (GLint x, GLint y, GLsizei width, GLsizei height); • void **glAlphaFunc** (GLenum func, GLclampf ref); • void **glStencilFunc** (GLenum func, GLint ref, GLuint mask); • void **glStencilOp** (GLenum fail, GLenum pass, GLenum zpass); • void **glDepthFunc** (GLenum func);	التحكم باختبار كل جزء على حدا جزء = fragment
• void **glBlendFunc** (GLenum sfactor, GLenum dfactor); • void **glLogicOp** (GLenum opcode);	التحكم بعملية المزج اللوني
• void **glClear** (GLbitfield mask);	محي كل أو بعض المخازن المؤقتة

• void **glClearAccum** (GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha); • void **glClearColor** (GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha); • void **glClearDepth** (GLclampd depth); • void **glClearIndex** (GLfloat c); • void **glClearStencil** (GLint s);	تحديد اللون الذي سيستخدم لمسح الصورة والعمق الذي ستنفذ عليه عملية المسح
• void **glDrawBuffer**(GLenum mode); • void **glIndexMask** (GLuint mask); • void **glColorMask** (GLboolean red, GLboolean green, GLboolean blue, GLboolean alpha); • void **glDepthMask** (GLboolean flag); • void **glStencilMask** (GLuint mask);	التحكم بإمكانية الكتابة على المخزن المؤقت
• void **glAccum** (GLenum op, GLfloat value);	إدارة المكدس
Evaluators	توفر طريقة لتحديد النقاط على المنحني أو السطح أو على جزء منهما باستخدام نقاط التحكم:
• void **glMap1**{fd}(GLenum target, TYPE u1, TYPE u2,GLint stride, GLint order, const TYPE *points); • void **glMap2**{fd}(GLenum target, TYPE u1, TYPE u2, Glint ustride, GLint uorder, TYPE v1, TYPE v2, GLint vstride, GLint vorder, const TYPE*points);	نستخدم one-dimensional evaluators لرسم منحني مثل منحني بيزييه مستخدمين فيه أربع نقاط تحكم نستخدم tow-dimensional evaluators لرسم سطح مثل سطح بيزييه مستخدمين أربعة نقاط تحكم.
• void **glMapGrid1**{fd}(GLint n, TYPE u1, TYPE u2); • void **glMapGrid2**{fd}(GLint un, TYPE u1, TYPE u2, GLint vn, TYPE v1, TYPE v2); • void **glEvalMesh1**(GLenum mode, GLint i1, Glint i2); • void **glEvalMesh2**(GLenum mode, GLint i1, GLint i2, GLint j1, GLint j2); • void **glEvalPoint1(**GLint i); • void **glEvalPoint2**(GLint i, GLint j);	توليد مجموعة من نقاط التحكم التي تنتمي لمجال نقاط الشكل الموضح بخطوط متصلة.

• void **glEvalCoord1**{fd}{v} (TYPE u); • void **glEvalCoord2**{fd}{v} (TYPE u, TYPE v);	تعمل بشكل مشابه تماماً لعمل التعليمة ()glVertex ويكون بارمتر هذه التعليمة عبارة عن إحدى قيم المتحول u الذي ينتمي للمجال [0,1].
• void **glGetMap**{idf}v (GLenum target, GLenum query, TYPE *v);	استعادة نقاط التحكم.
Selection and Feedback	
• GLint **glRenderMode** (GLenum mode); • void **glSelectBuffer** (GLsizei size, GLuint *buffer); • void **glFeedbackBuffer** (GLsizei size, GLenum type, GLfloat *buffer);	اختيار النموذج المستخدم، والمخزن المؤقت وإجراء عملية تغذية خلفية. تغذية خلفية = feedback
• void **glPassThrough** (GLfloat token);	تحديد الرمز من أجل تحديد نموذج التغذية الخلفية
• void **glInitNames** (void); • void **glLoadName** (GLuint name); • void **glPushName** (GLuint name); • void **glPopName** (void);	التحكم باسم المكدس من أجل عملية الاختيار
Display Lists قوائم الإظهار	
• void **glNewList** (GLuint list, GLenum mode); • void **glEndList** (void); • void **glDeleteLists** (GLuint list, GLsizei range);	إنشاء أو حذف قوائم الإظهار
• void **glCallList** (GLuint list); • void **glCallLists** (GLsizei n, GLenum type, const GLvoid *lists);	استدعاء قوائم الإظهار
• GLuint **glGenLists** (GLsizei range); • GLboolean **glIsList** (GLuint list); • void **glListBase** (GLuint base);	معالجة فهارس قوائم الإظهار
Modes and Execution النماذج والتنفيذ	
• void **glEnable** (GLenum cap); • void **glDisable** (GLenum cap); • GLboolean **glIsEnabled** (GLenum cap);	تفعيل وعدم تفعيل العمليات والاستفسار عن النموذج المستخدم
• void **glFinish** (void);	الانتظار حتى يتم تنفيذ جميع أوامر الـ OpenGL

• void **glFlush** (void);	تسبب تنفيذ أوامر الـــ OpenGL
• void **glHint** (GLenum target, GLenum mode);	إضافة بعض الملامح والصفات لعمليات الـــ OpenGL
State Queries	**الاستفسار عن الحالة**
• GLenum **glGetError** (void); • const GLubyte * **glGetString** (GLenum name);	الحصول على معلومات عن الأخطاء ، أو الاتصال الحالي بالـــ OpenGL
• void **glGetBooleanv** (GLenum pname, GLboolean *params); • void **glGetDoublev** (GLenum pname, GLdouble *params); • void **glGetFloatv** (GLenum pname, GLfloat *params); • void **glGetIntegerv** (GLenum pname, GLint *params);	الاستفسار عن متحولات الحالة
• void **glPushAttrib** (GLbitfield *mask*); • void **glPopAttrib** (void);	تخزين واسترجاع مجموعة متحولات الحالة

جدول بتعليمات مكتبة الـــ GLU:

Texture Images	**إكساء الصور**
• int **gluScaleImage** (GLenum format, GLint widthin, GLint heightin,GLenum typein, const void *datain, GLint widthout, GLint heightout, GLenum typeout, void *dataout);	تكبير أو تصغير الصور
• int **gluBuild1DMipmaps** (GLenum target, GLint components, GLint width, GLenum format, GLenum type, void *data); • int **gluBuild2DMipmaps** (GLenum target, GLint components, GLint width, GLint height, GLenum format, GLenum type, void *data);	توليد شكل من أشكال المقابلة حيث يتم حساب المظهر العام لصورة ما عن بعد ويستخدم الناتج كغلاف نسيجي

Coordinate Transformation	تحويل الإحداثيات:
إنشاء مصفوفات المنظور والرؤية	• void **gluOrtho2D** (GLdouble left, GLdouble right, GLdouble bottom,GLdouble top); • void **gluPerspective** (GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar); • void **gluPickMatrix** (GLdouble x, GLdouble y, GLdouble width, GLdouble height, GLint viewport[4]); • void **gluLookAt** (GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz);
تحويل من إحداثيات الغرض الثلاثية البعد إلى إحداثيات الشاشة الثنائية البعد	• int **gluProject** (GLdouble objx, GLdouble objy, GLdouble objz, const GLdouble modelMatrix[16], const GLdouble projMatrix[16], const GLint viewport[4], GLdouble *winx, GLdouble *winy, GLdouble *winz); • int **gluUnProject** (GLdouble winx, GLdouble winy, GLdouble winz, const GLdouble modelMatrix[16], const GLdouble projMatrix[16], const GLint viewport[4], GLdouble *objx, GLdouble *objy, GLdouble *objz);
تحويل من إحداثيات الغرض الثلاثية البعد إلى إحداثيات الشاشة الثنائية البعد	• int **gluProject** (GLdouble objx, GLdouble objy, GLdouble objz, const GLdouble modelMatrix[16], const GLdouble projMatrix[16], const GLint viewport[4], GLdouble *winx, GLdouble *winy, GLdouble *winz); • int **gluUnProject** (GLdouble winx, GLdouble winy, GLdouble winz, const GLdouble modelMatrix[16], const GLdouble projMatrix[16], const GLint viewport[4], GLdouble *objx, GLdouble *objy, GLdouble *objz);
Polygon Tessellation	المضلع الفسيفسائي:
التحكم بالكائنات الفسيفسائية	• GLUtriangulatorObj* **gluNewTess** (void); • void **gluTessCallback** (GLUtriangulatorObj *tobj, GLenum which, void (*fn)()); • void **gluDeleteTess** (GLUtriangulatorObj *tobj);

تحديد خصائص المضلع المدخل	• void **gluBeginPolygon** (GLUtriangulatorObj *tobj); • void **gluEndPolygon** (GLUtriangulatorObj *tobj); • void **gluNextContour** (GLUtriangulatorObj *tobj, GLenum type); • void **gluTessVertex** (GLUtriangulatorObj *tobj, GLdouble v[3], void *data);
Quadric Objects الكائنات من الدرجة الثانية:	
التحكم بالكائنات من الدرجة الثانية	• GLUquadricObj* **gluNewQuadric** (void); • void **gluDeleteQuadric** (GLUquadricObj *state); • void **gluQuadricCallback** (GLUquadricObj *qobj, GLenum which, void (*fn)());
التحكم بالأداء	• void **gluQuadricNormals** (GLUquadricObj *quadObject, GLenum normals); • void **gluQuadricTexture** (GLUquadricObj *quadObject, GLboolean textureCoords); • void **gluQuadricOrientation** (GLUquadricObj *quadObject, GLenum orientation); • void **gluQuadricDrawStyle** (GLUquadricObj *quadObject, GLenum drawStyle);
تحديد المواصفات البدائية للدرجة الثانية	• void **gluCylinder** (GLUquadricObj *qobj, GLdouble baseRadius, GLdouble topRadius, GLdouble height, GLint slices, GLint stacks); • void **gluDisk** (GLUquadricObj *qobj, GLdouble innerRadius, GLdouble outerRadius, GLint slices, GLint loops); • void **gluPartialDisk** (GLUquadricObj *qobj, GLdouble innerRadius, GLdouble outerRadius, GLint slices, GLint loops, GLdouble **startAngle**, GLdouble sweepAngle); • void **gluSphere** (GLUquadricObj *qobj, GLdouble radius, GLint slices, GLint stacks);

NURBS Curves and Surfaces	سطوح ومنحنيات NURBS :
• GLUnurbsObj* **gluNewNurbsRenderer** (void); • void **gluDeleteNurbsRenderer** (GLUnurbsObj *nobj); • void **gluNurbsCallback** (GLUnurbsObj *nobj, GLenum which, void (*fn)());	التحكم بكائنات NURBS
• void **gluBeginCurve** (GLUnurbsObj **nobj*); • void **gluEndCurve** (GLUnurbsObj **nobj*); • void **gluNurbsCurve** (GLUnurbsObj **nobj*, GLint *nknots*, GLfloat **knot*, GLint *stride*, GLfloat **ctlarray*, GLint *order*, GLenum *type*);	إنشاء منحنيات NURBS
• void **gluBeginSurface** (GLUnurbsObj *nobj);void gluEndSurface (GLUnurbsObj *nobj); • void **gluNurbsSurface** (GLUnurbsObj *nobj, GLint uknot_count, GLfloat *uknot, GLint vknot_count, GLfloat *vknot, GLint u_stride, GLint v_stride, GLfloat *ctlarray, GLint sorder, GLint torder, GLenum type);	إنشاء سطوح NURBS
• void **gluLoadSamplingMatrices** (GLUnurbsObj *nobj, const GLfloat modelMatrix[16], const GLfloat projMatrix[16], const GLint viewport[4]); • void **gluNurbsProperty** (GLUnurbsObj *nobj, GLenum property, GLfloat value); • void **gluGetNurbsProperty** (GLUnurbsObj *nobj, GLenum property, GLfloat *value);	التحكم بأداء NURBS

تعريف مجالات مزركشة	• void **gluBeginTrim** (GLUnurbsObj *nobj); • void **gluEndTrim** (GLUnurbsObj *nobj); • void **gluPwlCurve** (GLUnurbsObj *nobj, GLint count, GLfloat *array, GLint stride, GLenum type);
معالجة الأخطاء: **Error Handling**	
إعطاء رسالة خطأ يبين الخطأ في النص البرمجي للـــ OpenGL	• const GLubyte* gluErrorString (GLenum errorCode);

جدول بتعليمات مكتبة الـــ GLX:

التهيئة: **Initialization**	
تحديد فيما إذا كان الامتداد أو نطاق GLX (GLX extension) معرفاً على المخدم X .	• Bool **glXQueryExtension** (Display *dpy, int *errorBase, int *eventBase); • Bool **glXQueryVersion** (Display *dpy, int *major, int *minor);
تحديد الرؤية المرغوبة	• XVisualInfo* **glXChooseVisual** (Display *dpy, int screen, int *attribList); • int **glXGetConfig** (Display *dpy, XVisualInfo *vis, int attrib, int *value);
التحكم بالأداء: **Controlling Rendering**	
التحكم أو الاستفسار عن أداء بيئة الـــ OpenGL.	• GLXContext **glXCreateContext** (Display *dpy, XVisualInfo *vis, GLXContext shareList, Bool direct); • void **glXDestroyContext** (Display *dpy, GLXContext ctx); • void **glXCopyContext** (Display *dpy, GLXContext src, GLXContext dst, GLuint mask); • Bool **glXIsDirect** (Display *dpy, GLXContext ctx); • Bool **glXMakeCurrent** (Display *dpy, GLXDrawable draw, GLXContext ctx); GLXContext glXGetCurrentContext (void); • GLXDrawable **glXGetCurrentDrawable** (void);

• GLXPixmap **glXCreateGLXPixmap** (Display *dpy, XVisualInfo *vis, Pixmap pixmap); • void **glXDestroyGLXPixmap** (Display *dpy, GLXPixmap pix);	التحكم بأداء الشاشة
• void **glXWaitGL** (void); • void **glXWaitX** (void);	التنفيذ المتزامن
• void **glXSwapBuffers** (Display *dpy, Window window);	التبديل بين المخزن المؤقت الخلفي والأمامي
• void **glXUseXFont** (Font font, int first, int count, int listBase);	تحديد نوع الخط