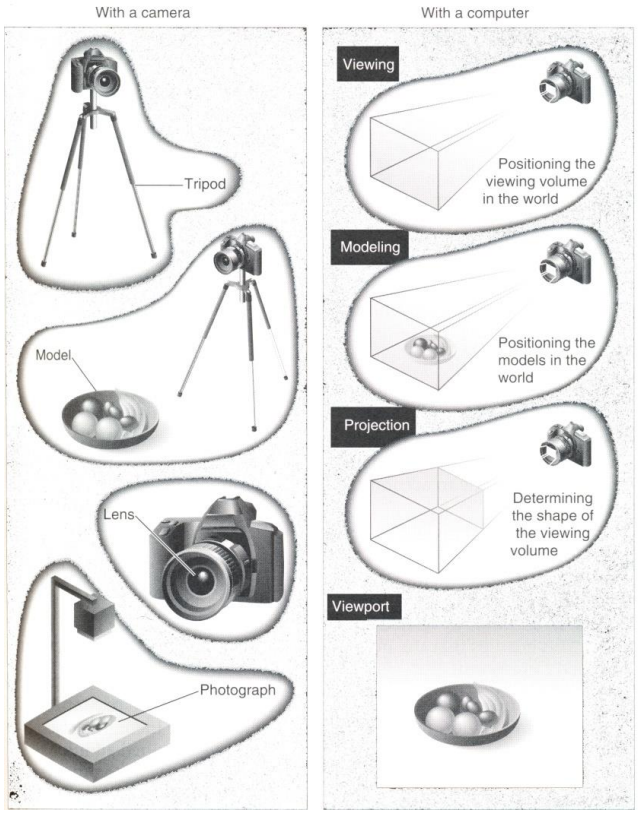
الجلسة الخامسة

مقدمة

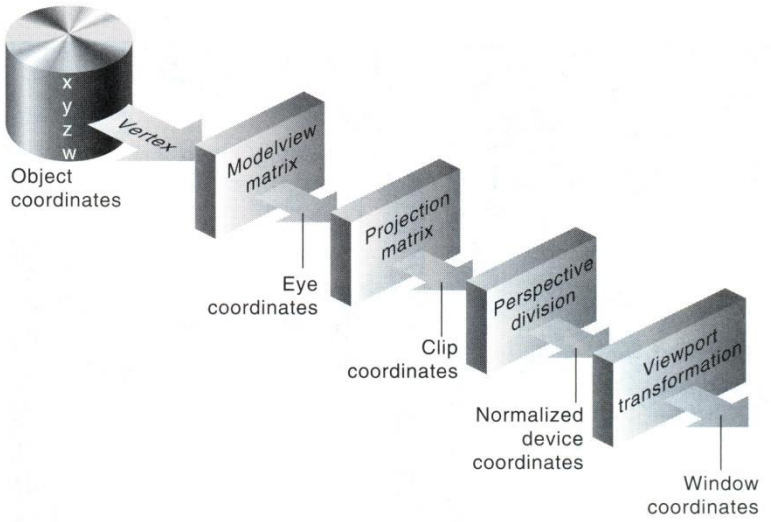
- سنتعلم في هذه الجلسة كيفية استخدام OpenGL للقيام بالوظائف التالية:
  - توضيع وتوجيه المجسمات في الفضاء ثلاثي الأبعاد.
  - ملائمة الموقع لعين الناظر.
- يجب تذكر أن هدف رسوميات الحاسب يتمثل في خلق صورة ثنائية البعد لأجسام ثلاثية البعد (لها بعدين فقط لأنها سترسم على الشاشة).

✓ **مبدأ عمل الكاميرا**

- يمكن تشبيه عملية التحويل في الحاسب للحصول على المشهد المطلوب رؤيته بعملية التقاط صورة بالكاميرا. والخطوات هي:
  - تثبيت حامل الكاميرا وتوجيه الكاميرا إلى المشهد (تحويلات viewing).
  - تنظيم المشهد ووضع عناصره في المكان المناسب (تحويلات modeling).
  - اختيار عدسة الكاميرا وتعديل التقريب والتبعيد zoom (تحويلات projection).
  - تحديد حجم الصورة النهائية (تحويلات viewport).
- بعد تنفيذ هذه الخطوات يمكن التقاط الصورة بواسطة الكاميرا وبشكل مشابه يمكن رسم المشهد على شاشة الحاسب.



- هناك ترتيب لهذه العمليات، فتحويلات viewing يجب أن تسبق تحويلات modeling في شيفرة البرنامج، أما تحويلات projection و viewport فيمكن تحديدها في أي مكان قبل حدوث الرسم. يبين الشكل التالي الترتيب الذي تنفذ فيه هذه العمليات على الحاسب.

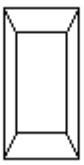


- لتحقيق تحويلات projection وmodeling وviewing سنبني مصفوفة M أبعادها 4x4 يتم ضربها بإحداثيات كل رأس v في المشهد لتحقيق التحويل المطلوب.

$V` = MV$

مثال:

- يرسم هذا المثال مكعباً يتم تحجيمه(تكبيره إلى الضعف وفق المحور y) ونقله مسافة بسيطة عكس المحور z عن طريق تحويل modeling. كذلك يتم تحديد تحويل إسقاط وتحويل viewport.



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


static void redraw(void);
int main(int argc, char **argv);
int main(int argc, char **argv)
{
    int h=300,w=300;
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(400,400);
    glutCreateWindow("draw Wire Cube");
    glutDisplayFunc(redraw);
```

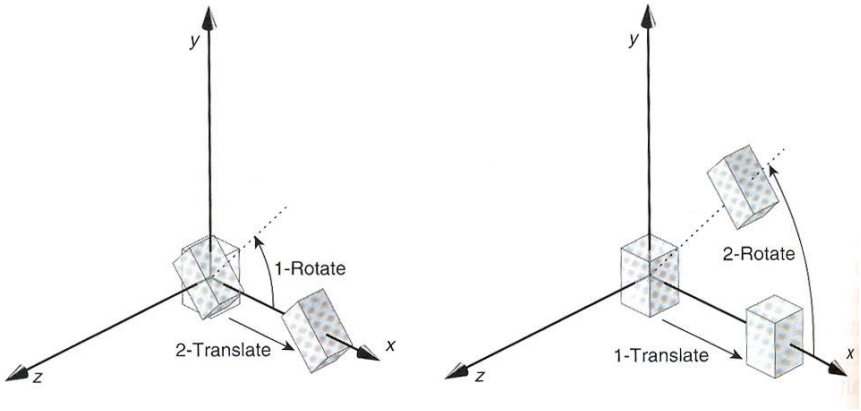
```
    glMatrixMode(GL_PROJECTION);/*  projection تعريف تحويل الاسقاط */
    gluPerspective(45,1.0,10.0,200.0);
    glMatrixMode(GL_MODELVIEW);
    glViewport (0, 0, w, h);      /*  viewport تعريف تحويل */
    glutMainLoop();
    return 0;
}
static void redraw(void)
{
   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glLoadIdentity ();
   glTranslatef (0.0, 0.0, -100.0);
   glScalef (1.0, 2.0, 1.0);      /* modeling تحويل */
   glutWireCube(25.0);    /*  رسم مكعب سلكي */
   glutSwapBuffers();
}
```

تحويلات viewing وmodeling

- قبل البدء بالتفاصيل تذكر أنه يجب استدعاء الأمر glMatrixMode() مع المعامل GL_MODELVIEW قبل تنفيذ تحويلات viewing أو modeling .

التفكير بالتحويلات :

- ترتيب التحويلات ضروري جداً:



- تحويلات viewing و modeling تمثل بمصفوفة 4x4. آخر تحويل يستدعى في برنامجك هو فعلياً أول تحويل يطبق على الرؤوس.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(N);                /* apply transformation N */
glMultMatrixf(M);                /* apply transformation M */
glMultMatrixf(L);                /* apply transformation L */
glBegin(GL_POINTS);
glVertex3f(v);                /* draw transformed vertex v */
glEnd();
```

- الناتج N(M(LV)).
- لنكتب الشيفرة البرمجية للشكل السابق على افتراض أن الدوران يتم أولاً ثم التحريك:

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

- تصدر أوامر التحويل viewing أولاً ثم modeling

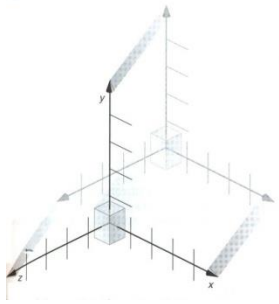
تحويلات modeling

- هناك ثلاثة تحويلات modeling وهي glTranslate*()، glRotate*()، glScale*().
- هذه التحويلات الثلاثة مكافئة لتشكيل مصفوفة النقل، والدوران، وتغيير الحجم ثم استدعاء glMultMatrix*() مع المصفوفة الملائمة كوسيط.

- **تحويل النقل Translate**

void glTranslate{fd}(TYPE*x*, TYPE *y*, TYPE*z*);

- تضرب المصفوفة الحالية بمصفوفة تنقل الجسم بمقدار قيم x,y,z كما هو مبين في الشكل التالي:



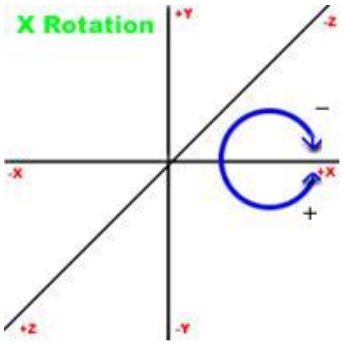
- مصفوفة النقل الناتجة T وهي :

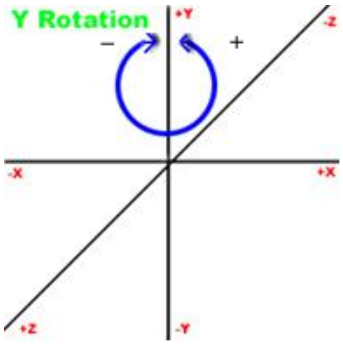
$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and } T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

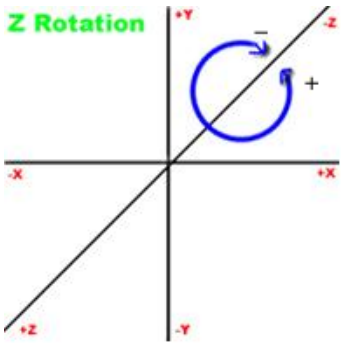
- **تحويل الدوران Rotate**

void **glRotate**{fd}(TYPE *angle*, TYPE *x*, TYPE *y*, TYPE *z*);

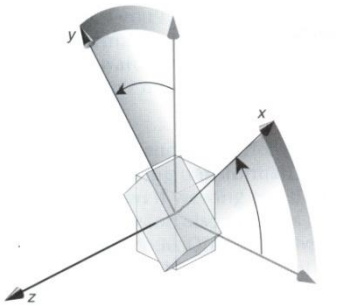
- يضرب المصفوفة الحالية بمصفوفة تدور الجسم بعكس عقارب الساعة(+) أو مع عقارب الساعة (-) حول محور معين وبزاوية تقدر بالدرجات.
- تبين الأشكال التالية الدوران الموجب والسالب للمحاور الثلاثة.







- يبين الشكل التالي تأثير الأمر glRotate(45.0,0.0,0.0,1.0):



- مصفوفة الدوران الناتجة R :

$$\text{glRotate*}(\alpha,\,1,\,0,\,0):\quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate*}(\alpha,\,0,\,1,\,0):\quad \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate*}(\alpha,\,0,\,0,\,1):\quad \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **تحويل تغيير الحجم Scale**

void **glScale**{fd}(**TYPE***x*, **TYPE** *y*, **TYPE***z*);

- يضرب المصفوفة الحالية بمصفوفة تكبر أو تمدد أو تقلص أو تعكس الجسم عبر المحاور. يضرب كل

إحداثي x,y,z لكل نقطة بالمعامل الموافق x,y,z. والجسم المرتبط يتمدد معها. يظهر الشكل التالي تأثير التحويل glScalef (2.0,-0.5,1.0)

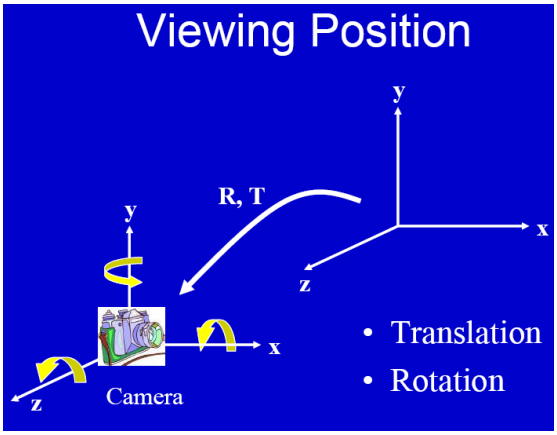
- مصفوفة تغيير الحجم الناتجة S :

$$S = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and } S^{-1} = \begin{bmatrix} \frac{1}{x} & 0 & 0 & 0 \\ 0 & \frac{1}{y} & 0 & 0 \\ 0 & 0 & \frac{1}{z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

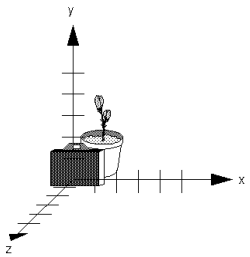
- القيم أكبر من 1 تكبير وأصغر من 1 تصغير. القيم 1- تعني عكس الجسم على المحور.

تحويلات viewing

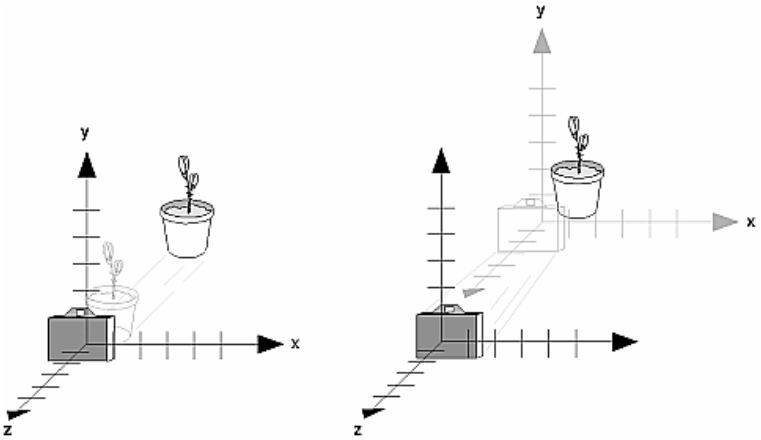
- تستخدم لتغيير موقع ووجهة نظر المشهد (الكاميرا). يشبه ذلك وضع أرجل الكاميرا وتوجيهها باتجاه الجسم. تتألف عادة تحويلات viewing من نقل ودوران.
- يكافئ تحويل modeling الذي يدور جسم عكس عقارب الساعة تحويل viewing الذي يدور الكاميرا مع عقارب الساعة.



- **استخدام glTranslate*() و glRotate*()**
  - توجه الكاميرا عادة نحو المحور z السالب. (نرى ظهر الكاميرا)



- تستطيع تحريك الكاميرا بعيداً عن الأجسام إلى الوراء وهذا له نفس تأثير تحريك الأجسام بعيداً عن الكاميرا للأمام. الأمر المستخدم glTranslatef(0.0,0.0,-5.0)

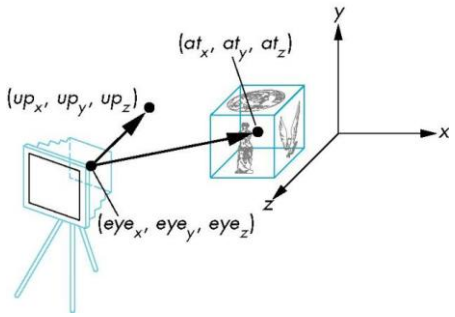


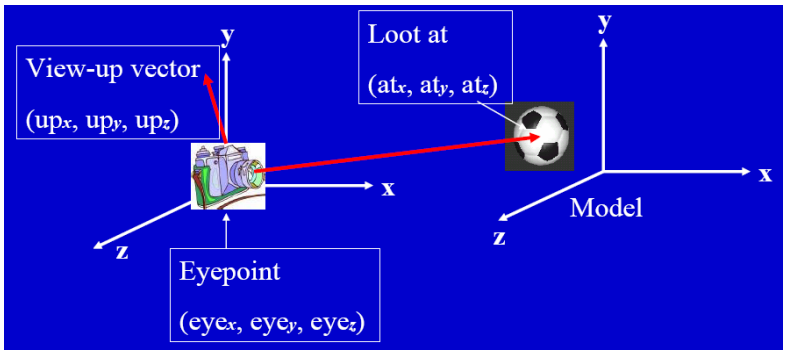
- إذا أردنا رؤية العناصر من الجانب يجب تدوير الجسم ثم تحريكه بعيداً عن الكاميرا حتى يمكن رؤية الجانب المطلوب من العنصر (تكتب الأوامر بعكس الترتيب التحريك ثم الدوران)

glTranslatef(0.0,0.0,-5.0) (جملة الاحداثيات المحلية)
glRotate(90.0,0.0,1.0,0.0);

- **التابع المسؤول عن الكاميرا:**

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



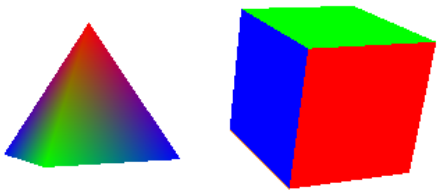


- **مثال:**

gluLookAt(0.0, 0.0, 50.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
gluLookAt(0.0, 0.0, -50.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

القسم العملي

تطبيق 1: تحويل المربع والمثلث من 2D إلى 3D (هرم ومكعب):



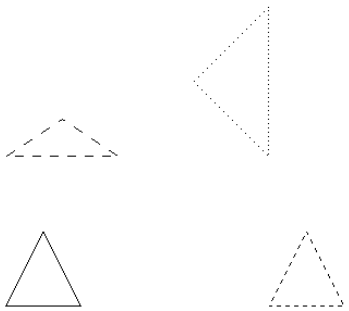
```
#include<GL/glut.h>
#include <stdlib.h>
#include<math.h>

static void redraw(void);
int main(int argc, char **argv)
{       glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
        glutInitWindowPosition(100,100);
        glutInitWindowSize(400,400);
        glutCreateWindow("draw payamid & cube");
        glutDisplayFunc(redraw);
        glMatrixMode(GL_PROJECTION);
        gluPerspective(45,1.0,10.0,200.0);
        glMatrixMode(GL_MODELVIEW);
        glutMainLoop();
        return 0;
 }
static void redraw(void)
{       glClearColor(1.0,1.0,1.0,0.0);
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        glTranslatef(-15.0f,0.0f,-100.0f);
        glRotatef(30,0.0f,1.0f,0.0f);                   // تدوير الهرم 30درجة حول المحور y //
        glBegin(GL_TRIANGLES);                          // بدء رسم المثلث //
                glColor3f(1.0f,0.0f,0.0f);          // أحمر
                glVertex3f( 0.0f, 10.0f, 0.0f);             // النقطة العلوية في المثلث –الواجهة الأمامية للهرم //
                glColor3f(0.0f,10.0f,0.0f);         // أخضر
                glVertex3f(-10.0f,-10.0f, 10.0f);   // النقطة اليسارية في المثلث –الواجهة الأمامية للهرم //
                glColor3f(0.0f,0.0f,10.0f);         // أزرق
                glVertex3f( 10.0f,-10.0f, 10.0f);   // النقطة اليمينية في المثلث –الواجهة الأمامية للهرم //
                glColor3f(10.0f,0.0f,0.0f);         // أحمر
                glVertex3f( 0.0f, 10.0f, 0.0f);             // النقطة العلوية في المثلث –الواجهة اليمينية للهرم //
                glColor3f(0.0f,0.0f,1.0f);          // أزرق
                glVertex3f( 10.0f,-10.0f, 10.0f);   // النقطة اليسارية في المثلث –الواجهة اليمينية للهرم //
                glColor3f(0.0f,10.0f,0.0f);         // أخضر
                glVertex3f( 10.0f,-10.0f, -10.0f);  // النقطة اليمينية في المثلث –الواجهة اليمينية للهرم //
                glColor3f(1.0f,0.0f,0.0f);          // أحمر
                glVertex3f( 0.0f, 10.0f, 0.0f);             // النقطة العلوية في المثلث –الواجهة الخلفية للهرم //
                glColor3f(0.0f,1.0f,0.0f);          // أخضر
                glVertex3f( 10.0f,-10.0f, -10.0f);  // النقطة اليسارية في المثلث –الواجهة الخلفية للهرم //
                glColor3f(0.0f,0.0f,1.0f);          // أزرق
```

```
        glVertex3f(-10.0f,-10.0f, -10.0f);   // النقطة اليمينية في المثلث –الواجهة الخلفية للهرم
        glColor3f(1.0f,0.0f,0.0f);           // أحمر
        glVertex3f( 0.0f, 10.0f, 0.0f);              // النقطة العلوية في المثلث –الواجهة اليسارية للهرم
        glColor3f(0.0f,0.0f,1.0f);           // أزرق
        glVertex3f(-10.0f,-10.0f,-10.0f);    // النقطة اليسارية في المثلث –الواجهة اليسارية للهرم
        glColor3f(0.0f,1.0f,0.0f);           // أخضر
        glVertex3f(-10.0f,-10.0f, 10.0f);    // النقطة اليمينية في المثلث –الواجهة اليسارية للهرم
    glEnd();                                 // نهاية رسم الهرم
    glLoadIdentity();                                // اعادة تهيئة مصفوفة التحويلات
    glTranslatef(20.0f,0.0f,-100.0f);
    glRotatef(30,1.0f,1.0f,0.0f);
    glBegin(GL_QUADS);              // رسم المكعب
        glColor3f(0.0f,1.0f,0.0f);           // أخضر
        glVertex3f( 10.0f, 10.0f,-10.0f);    // النقطة العلوية اليمينية في المربع–الواجهة العلوية للمكعب
        glVertex3f(-10.0f, 10.0f,-10.0f);    // النقطة العلوية اليسارية في المربع–الواجهة العلوية للمكعب
        glVertex3f(-10.0f, 10.0f, 10.0f);    // النقطة السفلية اليسارية في المربع–الواجهة العلوية للمكعب
        glVertex3f( 10.0f, 10.0f, 10.0f);    // النقطة السفلية اليمينية في المربع–الواجهة العلوية للمكعب
        glColor3f(1.0f,0.5f,0.0f);           // برتقالي
        glVertex3f( 10.0f,-10.0f, 10.0f);    // النقطة العلوية اليمينية في المربع–الواجهة السفلية للمكعب
        glVertex3f(-10.0f,-10.0f, 10.0f);    // النقطة العلوية اليسارية في المربع–الواجهة السفلية للمكعب
        glVertex3f(-10.0f,-10.0f,-10.0f);    // النقطة السفلية اليسارية في المربع–الواجهة السفلية للمكعب
        glVertex3f( 10.0f,-10.0f,-10.0f);    // النقطة السفلية اليمينية في المربع–الواجهة السفلية للمكعب
        glColor3f(1.0f,0.0f,0.0f);           // أحمر
        glVertex3f( 10.0f, 10.0f, 10.0f);    // النقطة العلوية اليمينية في المربع–الواجهة الأمامية للمكعب
        glVertex3f(-10.0f, 10.0f, 10.0f);    // النقطة العلوية اليسارية في المربع–الواجهة الأمامية للمكعب
        glVertex3f(-10.0f,-10.0f, 10.0f);    // النقطة السفلية اليسارية في المربع–الواجهة الأمامية للمكعب
        glVertex3f( 10.0f,-10.0f, 10.0f);    // النقطة السفلية اليمينية في المربع–الواجهة الأمامية للمكعب
        glColor3f(1.0f,1.0f,0.0f);           // أصفر
        glVertex3f( 10.0f,-10.0f,-10.0f);    // النقطة العلوية اليمينية في المربع–الواجهة الخلفية للمكعب
        glVertex3f(-10.0f,-10.0f,-10.0f);    // النقطة العلوية اليسارية في المربع–الواجهة الخلفية للمكعب
        glVertex3f(-10.0f, 10.0f,-10.0f);    // النقطة السفلية اليسارية في المربع–الواجهة الخلفية للمكعب
        glVertex3f( 10.0f, 10.0f,-10.0f);    // النقطة السفلية اليمينية في المربع–الواجهة الخلفية للمكعب
        glColor3f(0.0f,0.0f,1.0f);           // أزرق
        glVertex3f(-10.0f, 10.0f, 10.0f);    // النقطة العلوية اليمينية في المربع–الواجهة اليسارية للمكعب
        glVertex3f(-10.0f, 10.0f,-10.0f);    // النقطة العلوية اليسارية في المربع–الواجهة اليسارية للمكعب
        glVertex3f(-10.0f,-10.0f,-10.0f);    // النقطة السفلية اليسارية في المربع–الواجهة اليسارية للمكعب
        glVertex3f(-10.0f,-10.0f, 10.0f);    // النقطة السفلية اليمينية في المربع–الواجهة اليسارية للمكعب
        glColor3f(1.0f,0.0f,1.0f);           // بنفسجي
        glVertex3f( 10.0f, 10.0f,-10.0f);    // النقطة العلوية اليمينية في المربع–الواجهة اليمينية للمكعب
        glVertex3f( 10.0f, 10.0f, 10.0f);    // النقطة العلوية اليسارية في المربع–الواجهة اليمينية للمكعب
        glVertex3f( 10.0f,-10.0f, 10.0f);    // النقطة السفلية اليسارية في المربع–الواجهة اليمينية للمكعب
        glVertex3f( 10.0f,-10.0f,-10.0f);    // النقطة السفلية اليمينية في المربع–الواجهة اليمينية للمكعب
```

```
        glEnd();                                    // نهاية رسم المكعب
        glutSwapBuffers();
}
```

- تعديل1: يطلب الحصول على أشكال صحيحة ثلاثية البعد.
- تعديل2: يطلب إضافة قاعدة للهرم.
- تعديل3: يطلب رؤية المكعب والهرم من زوايا رؤية (كاميرا) مختلفة.

تطبيق 2: استخدام تحويلات modeling لرسم المثلثات التالية:



```
static void redraw(void)
{
        float x1,x2,x3,y1,y2,y3;
#define draw_triangle(x1,y1,x2,y2,x3,y3) glBegin(GL_LINE_LOOP);\
      glVertex2f(x1,y1);glVertex2f(x2,y2);glVertex2f(x3,y3);glEnd();
     glClearColor(1,1,1,0);
     glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    glEnable(GL_LINE_STIPPLE);
        //رسم المثلث اليساري السفلي
        glLineStipple(1, 0xFFFF);
    glLoadIdentity ();
    glTranslatef (-15.0, 0.0, -70.0);
    draw_triangle(-5.0,-10.0,-10.0,-20.0,0.0,-20.0);
        //رسم المثلث اليميني  السفلي
        glLineStipple(1, 0xF0F0);
        glLoadIdentity();
        glTranslatef(10.0, 0.0, -70.0);
        draw_triangle(5.0,-10.0,0.0,-20.0,10.0,-20.0);
        //رسم المثلث اليساري العلوي
        glLineStipple(1, 0xF00F);
        glLoadIdentity();
        glTranslatef(-10.0, 0.0, -70.0);
        glScalef(1.5, 0.5, 1.0);
        draw_triangle(-5.,10.0,-10.0,0.0,0.0,0.0);
        //رسم المثلث اليميني العلوي
        glLineStipple(1, 0x8888);
        glLoadIdentity();
        glTranslatef(20.0, 0.0, -70.0);
        glRotatef (90.0, 0.0, 0.0, 1.0);
        draw_triangle (10.0,20.0,0.0,10.0,20.0,10.0);
        glDisable (GL_LINE_STIPPLE);
        glutSwapBuffers();
}
```

تعديل:

إذا علمت أنه يجب استخدام تعليمة glLoadIdentity مرة واحدة فقط في البرنامج السابق، عدل ما يلزم لرسم الأشكال الموضحة أعلاه.