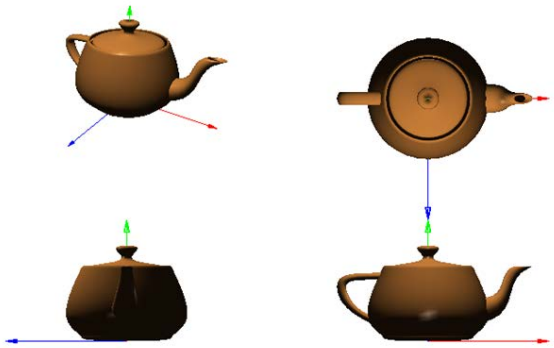
Cámara

Seminario ISGI (S4)



R. Vivó

R. Vivó

Cámara

Seminario ISGI (S4)

Este seminario explica cómo elegir una cámara para visualizar la escena, qué parámetros son relevantes para su definición y cuál es su impacto en las transformaciones que sufren los objetos. También se describe cómo activar la visibilidad para que los objetos más cercanos oculten a los que están detrás dando mayor sensación de profundidad.

Tipos de cámaras

La cámara es un elemento de la escena que nos permite decidir cómo y desde dónde vamos a ver los objetos. En OpenGL podemos usar dos tipos de cámara: la ortográfica y la perspectiva.

La cámara ortográfica produce fotografías de la escena donde dos líneas que son paralelas en 3D siguen siendo paralelas en la foto. Este tipo de fotografía se usa en aplicaciones técnicas, como los planos, pues las distancias medidas en la foto corresponden a las reales multiplicadas por un factor de escala, que es el mismo dada una dirección principal.

La cámara perspectiva se usa para simular la visión real del ojo, pues los objetos que se encuentran más alejados se representan más pequeños. En la foto no se pueden medir distancias ni las líneas conservan su paralelismo pero la sensación es más real que en la fotografía ortográfica.

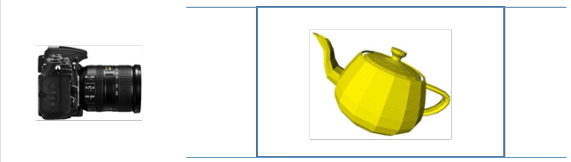


Figura 1. Cámara ortográfica

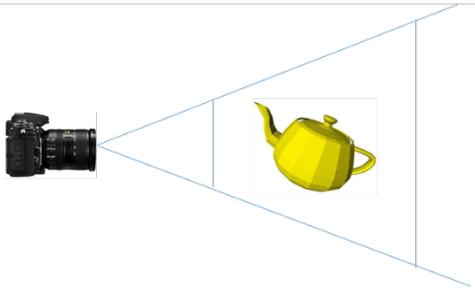


Figura 2. Cámara perspectiva

Al igual que se hace con una cámara real, para definir el tipo de cámara es necesario concretar ciertos parámetros como el campo de visión dado por el tipo de objetivo (milímetros) o la relación entre el ancho y el alto de la fotografía. Por ejemplo, un objetivo de gran angular produce un campo de visión mayor que un teleobjetivo. Además de estos parámetros hay otros que son específicos de la cámara virtual como si es ortográfica o perspectiva, o cuáles son los planos cercano y lejano que limitan la foto. Otros valores de la cámara real como la profundidad de campo, la apertura del objetivo (cantidad de luz que entra) o la velocidad de disparo no se tienen en cuenta en la cámara sintética.

Una vez decidida cuál es la cámara que se va usar es necesario situarla en la escena. Fundamentalmente se trata de indicar dónde se sitúa la cámara, hacia dónde está mirando y cuál es su orientación (por ejemplo podemos poner la cámara horizontal o vertical).

A continuación vemos primero cómo definir la cámara, cómo procurar que la foto salga proporcionada y cómo situar la cámara en la escena usando órdenes de OpenGL.

Volumen de la vista

El volumen de la vista es la región del espacio 3D que saldrá fotografiada. La forma que tenga este volumen determina el tipo de cámara.

Para la cámara ortográfica hay que definir un volumen que es un ortoedro. Para esto se usa la orden de OpenGL:

```
glOrtho(izquierda, derecha, abajo, arriba, cerca, lejos)
```

donde todos los parámetros son números reales que indican los planos que delimitan el ortoedro. Los planos son paralelos a los planos principales del sistema de coordenadas del observador (cámara). Así, por ejemplo, si la cámara mira hacia el eje -***Z*** y la horizontal y la vertical son los ejes ***X*** e ***Y*** respectivamente, el volumen es una caja de caras paralelas a los planos principales de la escena.

Si no se define ninguna cámara, la que hay por defecto es una cámara ortográfica definida así:

```
glOrtho(-1,1,-1,1,-1,1)
```

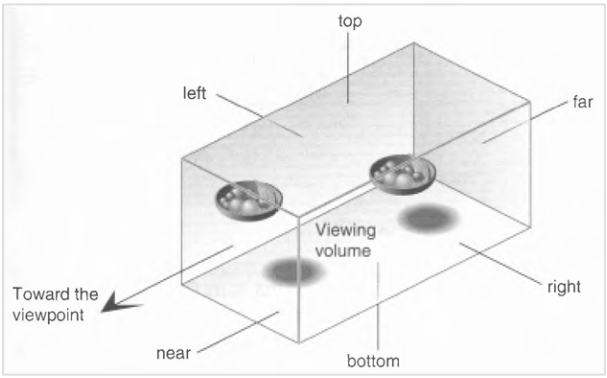


Figura 3. Volumen de la vista ortográfica

Para la cámara perspectiva hay que definir un volumen que es una pirámide truncada. Para esto se usa la orden de la librería GLU (anexa a la OpenGL):

```
gluPerspective(fovy, razon, cerca, lejos)
```

donde todos los parámetros son números reales. El primero indica el ángulo vertical en grados que tiene la pirámide de visión. El segundo indica la relación entre el ancho y el alto de la base de la pirámide. El tercero y el cuarto corresponden a la distancia donde se debe truncar la pirámide y a la distancia a la que está la base, respectivamente. Las distancias se miden desde la posición de la cámara que es el vértice de la pirámide. La Figura 4 muestra el volumen de la vista perspectiva.

Al definirse el volumen de la vista queda también definida la matriz que se usa en la transformación proyectiva de la Figura 5. OpenGL almacena esta matriz en la **matriz de la proyección** a la que se nombra como `GL_PROJECTION`. Es necesario indicar que se va a trabajar con esta matriz, antes de definir el volumen de la vista, para que se almacene en el sitio correcto:

```
glMatrixMode(GL_PROJECTION)
```

y cargar la identidad con `glLoadIdentity()` antes de acumular nuestra propia matriz de proyección con `glOrtho()` o `gluPerspective()`.

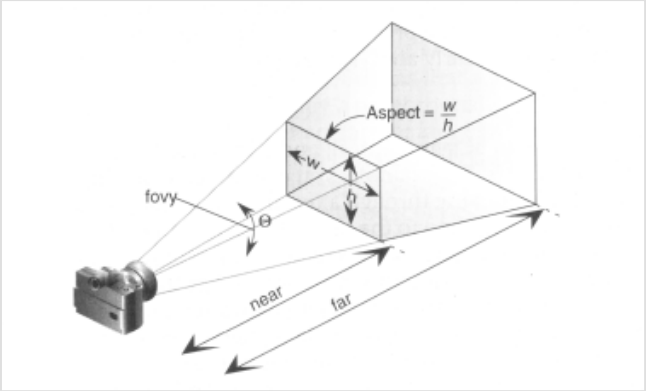


Figura 4. Volumen de la vista perspectiva

Al igual que sucede con las transformaciones de los objetos almacenadas en la *modelview*, la *projection* se maneja en una pila, siendo la matriz superior de la pila la que se aplica en cada momento. Esto permite, por ejemplo, cambiar el tipo de cámara para sacar diferentes fotografías del mismo objeto haciendo uso de las órdenes `glPushMatrix()` y `glPopMatrix()` estando seleccionada la `GL_PROJECTION`.

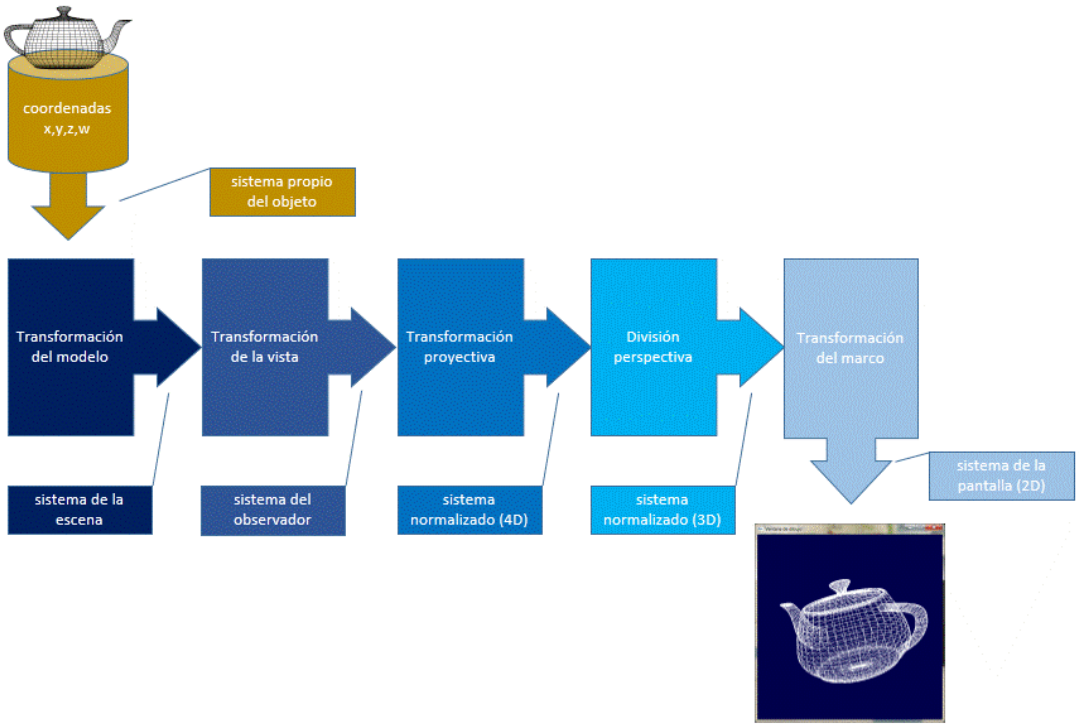


Figura 5. Cadena de transformaciones

Los volúmenes ortográfico y piramidal son rectos, es decir, la visual que pasa por el centro de la foto es perpendicular a ella. Aunque es posible generar volúmenes oblicuos (por ejemplo al mirar por una ventana que está arriba y a nuestra derecha) no son de mucha utilidad y complican la comprensión de la imagen.

El marco y la razón de aspecto

Una vez tomada la fotografía hay que llevarla a un soporte físico como un papel o una pantalla para verla. Si las dimensiones del soporte no tienen la misma razón de aspecto que la foto, saldrá deformada.

En la cámara ortográfica la razón de aspecto viene dada por la relación (derecha-izquierda)/(arriba-abajo) mientras que en la perspectiva se da directamente como parámetro `-razon-`.

Por tanto, para que no haya deformación, podemos seguir dos estrategias:

1. Ajustar la razón de aspecto de la foto a la del soporte físico
2. Ajustar la razón de aspecto del soporte físico a la de la foto

En la tarea de generar una imagen en pantalla el soporte físico es un área rectangular de pixeles. A esta área la llamamos **marco**. El marco es, por tanto, donde ponemos la foto. Para fijar las dimensiones del marco se usa la orden de OpenGL:

```
glViewport(i , j, ancho, alto)
```

donde `i` y `j` son las coordenadas del pixel inferior izquierdo del marco en la ventana de dibujo y `ancho` y `alto` su dimensión horizontal y vertical en pixeles. Hay que hacer notar que el píxel (0,0) es el inferior izquierdo de la ventana de dibujo para lo que al marco se refiere.

La llamada a esta función genera directamente la **matriz de transformación del marco** de la Figura 5 que es única y no se maneja con pilas como las anteriores.

Volviendo al problema de la deformación analicemos qué nos conviene. Parece lógico plantear los siguientes requisitos:

a) Dada el área de dibujo de, digamos, 500x400 pixeles el marco debería utilizarlos todos, es decir, empezar en el 0,0 y tener 500 de ancho y 400 de alto. Así aprovechamos toda la ventana.
b) La decisión de qué fotografiar debe tomarse, en la medida de lo posible, al tomar la foto, no al colocarla en el marco. Esto quiere decir que si el área de dibujo es más grande y el marco se ajusta a su área total, la foto se verá más grande.

Ante estos requisitos parece conveniente: a) ajustar el marco al área de dibujo; y b) elegir libremente el volumen de la vista siempre que su razón de aspecto coincida con la del marco. Veamos cómo saber entonces las dimensiones del área de dibujo.

Redimensionamiento del área de dibujo

La librería GLUT maneja el evento de redimensión del área de dibujo. Este evento sucede cuando se crea la ventana o cuando el usuario estira de los cantos. La manera de procesar este evento es mediante una *callback* que se registra así:

```
glutReshapeFunc(onReshape)
```

donde `onReshape` es, por ejemplo, el nombre de la función a la que se llama cuando sucede el evento. Esta función debe declararse con dos parámetros así:

```
void onReshape(int w, int h)
```

donde `w` y `h` son las dimensiones de la nueva ventana en pixeles que trae como mensaje el evento.

Ejercicio S4E01: Construir una función onReshape() para que haya isotropía, según la cámara elegida, ajustando el marco a la totalidad área de dibujo variando la razón de aspecto de la vista.

Ejercicio S4E02: Construir una función onReshape() para que haya isotropía, según la cámara elegida, encajando lo mejor posible el marco en el área de dibujo sin variar la razón de aspecto de la vista.

Situando la cámara

Una vez decidida la cámara a utilizar y los parámetros de configuración de su volumen de la vista, es la hora de colocarla en la escena. Para ello hay que determinar:

1. La posición de la cámara. En la proyección perspectiva la posición de la cámara es el punto donde se sitúa el observador. En la proyección ortográfica, donde el observador está en el infinito, la posición de la cámara es el punto a partir del cual se miden las distancias de los planos cercano y lejano del volumen de la vista. En ambos casos la posición de la cámara sirve para fijar el origen del sistema de coordenadas de la vista.
2. La dirección central hacia donde apunta la cámara. Lo más fácil es dar un punto del espacio que queremos que salga en el centro de la foto. Esto determina la visual central como la línea que va desde la posición de la cámara hasta ese punto. En la proyección perspectiva la visual central pasa por el centro de la pirámide y en la ortográfica por el centro del ortoedro.
3. La vertical de la cámara. Es el vector del espacio que, al ser proyectado -fotografiado-, queremos que salga paralelo al canto vertical de la foto. En realidad hay infinitos vectores que cumplen esta condición basta imaginar un papel con vectores dibujados en cualquier dirección que se coloca de tal manera que sólo vemos su canto paralelo a la vertical de la foto (ver Figura 6). De entre todos estos vectores daremos el que más fácil sea de identificar.

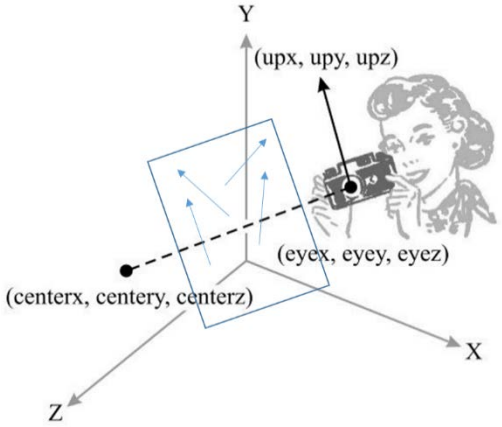


Figura 6. Colocando la cámara

Para colocar la cámara en OpenGL se usa una orden de la librería GLU:

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

donde *(eyex,eyey,eyez)* son las coordenadas reales del punto de vista -posición de la cámara-, *(centerx,centery,centerz)* son las coordenadas reales del punto central al que se mira y *(upx,upy,upz)* son las componentes reales de cualquier vector que se proyecte vertical.

La llamada a `glLookAt()` construye la **matriz de la vista** que en OpenGL viene unida a la del modelo en la matriz *modelview* (ver Figura 5). Para que la composición de las dos matrices se haga en la secuencia correcta hay que:

1. activar la *modelview* con `glMatrixMode()`
2. cargar la matriz identidad para empezar sin transformaciones

3. colocar la cámara
4. transformar los objetos
5. dibujar

De esta manera, la matriz de la vista se compone por la izquierda con la matriz del modelo para que sea esta última la primera que se aplique a los objetos.

Ejercicio S4E03: Dibujar una tetera en perspectiva en el punto (2,0,0) vista desde arriba y centrada en la pantalla. ¿Qué efecto tiene modificar la vertical de la foto?

Visibilidad

La presencia de un observador en la escena establece un orden en la visibilidad de los objetos. Aquellos que estén más cercanos puede que tapen a los más alejados. El proceso por el que se determina qué objetos o qué partes de ellos se ven se llama **cálculo de la visibilidad**.

El cálculo de la visibilidad lo hace OpenGL como uno de los procesos en su tubería gráfica. Simplemente hay que activarla siguiendo estos pasos:

1. Disponer un *buffer* para el cálculo de la visibilidad. Indicaremos el uso de este *buffer* añadiendo `GLUT_DEPTH` al registro de inicio así:

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

2. En la *callback* de atención al evento de redibujado limpiamos el *buffer* de visibilidad al igual que hacemos con el de color:

   ```
   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
   ```

3. Activamos o desactivamos el cálculo de la visibilidad con la constante `GL_DEPTH_TEST`:

   ```
   glEnable(GL_DEPTH_TEST)
   ```

   ```
   glDisable(GL_DEPTH_TEST)
   ```

Además de lo anterior, para que los objetos más cercanos oculten a los más alejados deben ser dibujados como polígonos, usando los diferentes tipos de primitivas poligonales que ofrece OpenGL, por ejemplo `GL_POLYGON`.

Ejercicio S4E04: ¿Qué ocurre cuando se dibuja primero un objeto más cercano y después uno más alejado que está parcialmente tapado si se deshabilita el test de visibilidad? Probar usando glutSolidTeapot().

Ejercicio S4E01: Construir una función onReshape() para que haya isotropía, según la cámara elegida, ajustando el marco a la totalidad área de dibujo variando la razón de aspecto de la vista.

```
/*************************************************
ISGI::Isotropia de la vista
Roberto Vivo', 2013 (v1.0)

Dibujo de una esfera donde la vista se ajusta al marco

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S4E01::Isotropia Vista"

#include <iostream>                             // Biblioteca de entrada salida
#include <gl\freeglut.h>                        // Biblioteca grafica

void display()
// Funcion de atencion al dibujo
{
  glClear(GL_COLOR_BUFFER_BIT);                 // Borra la pantalla
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  gluLookAt(0,0,5,0,0,0,0,1,0);                 // Situa la camara

  glutWireSphere(1.0,20,20);                    // Dibuja la esfera

  glFlush();                                    // Finaliza el dibujo
}

void reshape(GLint w, GLint h)
// Funcion de atencion al redimensionamiento
{
        // Usamos toda el area de dibujo
        glViewport(0,0,w,h);

        // Definimos la camara (matriz de proyeccion)
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

        float razon = (float) w / h;

        /* CAMARA ORTOGRAFICA
           Ajustamos la vista a la dimenson más pequenya del viewport para
           poder ver la totalidad de la ventana del mundo real (2x2)
        if(w<h)
                glOrtho(-1,1,-1/razon,1/razon, 0,10);
        else
                glOrtho(-1*razon,1*razon,-1,1, 0,10); */

        /* CAMARA PERSPECTIVA
           La razon de aspecto se pasa directamente a la camara perspectiva
           Como damos fijo el angulovertical, el tamanyo del dibujo solo se
           modifica cuando variamos la altura del viewport */
        gluPerspective(45,razon,1,10);
}

void main(int argc, char** argv)
// Programa principal
{
  glutInit(&argc, argv);                                // Inicializacion de GLUT
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);            // Alta de buffers a usar
  glutInitWindowSize(400,400);                          // Tamanyo inicial de la ventana
  glutCreateWindow(PROYECTO);                           // Creacion de la ventana con su titulo
  std::cout << PROYECTO << " running" << std::endl;     // Mensaje por consola
  glutDisplayFunc(display);                             // Alta de la funcion de atencion a display
  glutReshapeFunc(reshape);                             // Alta de la funcion de atencion a reshape
  glutMainLoop();                                       // Puesta en marcha del programa
}
```

Ejercicio S4E02: Construir una función onReshape() para que haya isotropía, según la cámara elegida, encajando lo mejor posible el marco en el área de dibujo sin variar la razón de aspecto de la vista.

```
void reshape(GLint w, GLint h)
// Funcion de atencion al redimensionamiento
{
        // Razon de aspecto de la vista
        static const float razon = 2.0;                 // a/b = w'/h'

        // Razon de aspecto del area de dibujo
        float razonAD= float(w)/h;

        float wp,hp;                                    // w',h'
        /* Centramos un viewport con la misma razon de la vista.
           Si el area tiene razon menor la vista se ajusta en horizontal (w)
           recortando el viewport por arriba y por abajo.
           Si el area tiene mayor razon que la vista se ajusta en vertical (h)
           y se recorta por la izquierda y la derecha. */
        if(razonAD<razon){
                wp= float(w);
                hp= wp/razon;
                glViewport(0,int(h/2.0-hp/2.0),w,int(hp));
        }
        else{
                hp= float(h);
                wp= hp*razon;
                glViewport(int(w/2.0-wp/2.0),0,int(wp),h);
        }

        // Definimos la camara (matriz de proyeccion)
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

        /* CAMARA ORTOGRAFICA
        glOrtho(-1,1,-1/razon,1/razon, 0,10);*/

        /* CAMARA PERSPECTIVA */
        gluPerspective(45,razon,1,10);
}
```

Ejercicio S4E03: Dibujar una tetera en perspectiva en el punto (2,0,0) vista desde arriba y centrada en la pantalla. ¿Qué efecto tiene modificar la vertical de la foto?

```
/************************************************
ISGI::Vista cenital
Roberto Vivo', 2013 (v1.0)

Dibujo de una teter vista desde arriba

Dependencias:
+GLUT
************************************************/
#define PROYECTO "ISGI::S4E03::Vista Cenital"

#include <iostream>                                     // Biblioteca de entrada salida
#include <gl\freeglut.h>                                // Biblioteca grafica

void display()
// Funcion de atencion al dibujo
{
  glClear(GL_COLOR_BUFFER_BIT);                         // Borra la pantalla
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  gluLookAt(2,5,0,2,0,0, 0,0,-1);                       // Situa la camara

  glTranslatef(2,0,0);
  glutWireTeapot(1.0);                                  // Dibuja la tetera

  glFlush();                                            // Finaliza el dibujo
}

void reshape(GLint w, GLint h)
// Funcion de atencion al redimensionamiento
{
        // Usamos toda el area de dibujo
        glViewport(0,0,w,h);

        // Definimos la camara (matriz de proyeccion)
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

        float razon = (float) w / h;
        gluPerspective(45,razon,1,10);
}
```

Ejercicio S4E04: ¿Qué ocurre cuando se dibuja primero un objeto más cercano y después uno más alejado que está parcialmente tapado si se deshabilita el test de visibilidad? Probar usando glutSolidTeapot().

```
/*************************************************
ISGI::Visibilidad
Roberto Vivo', 2013 (v1.0)

Dibujo de una teter vista desde arriba

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S4E04::Visibilidad"

#include <iostream>                                     // Biblioteca de entrada salida
#include <gl\freeglut.h>                                // Biblioteca grafica

void display()
// Funcion de atencion al dibujo
{
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);     // Borra la pantalla y el Z-buffer
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  gluLookAt(0,0,5,0,0,0,0,1,0);                         // Situa la camara

  //glEnable(GL_DEPTH_TEST);

  glColor3f(1,0,0);
  glutWireTeapot(0.5);                                  // Dibuja la tetera cercana roja

  glTranslatef(1,0,-4);
  glColor3f(0,0,1);
  glutSolidTeapot(0.5);                                 // Dibuja la tetera lejana azul

  glFlush();                                            // Finaliza el dibujo
}

void reshape(GLint w, GLint h)
// Funcion de atencion al redimensionamiento
{
        // Usamos toda el area de dibujo
        glViewport(0,0,w,h);

        // Definimos la camara (matriz de proyeccion)
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

        float razon = (float) w / h;
        gluPerspective(45,razon,1,10);
}

void main(int argc, char** argv)
// Programa principal
{
  glutInit(&argc, argv);                                // Inicializacion de GLUT
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH); // Alta de buffers a usar
  glutInitWindowSize(400,400);                          // Tamanyo inicial de la ventana
  glutCreateWindow(PROYECTO);                           // Creacion de la ventana con su titulo
  std::cout << PROYECTO << " running" << std::endl;     // Mensaje por consola
  glutDisplayFunc(display);                             // Alta de la funcion de atencion a display
  glutReshapeFunc(reshape);                             // Alta de la funcion de atencion a reshape
  glutMainLoop();                                       // Puesta en marcha del programa
}
```