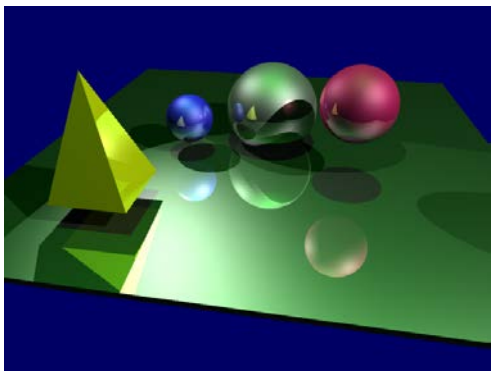
02/09/2016

Iluminación

Seminario ISGI (S7)



R. Vivó

R. Vivó

Iluminación

Seminario ISGI (S7)

La iluminación confiere a los objetos una apariencia realista y es una etapa fundamental en la formación de la imagen. La iluminación depende de las características de las fuentes de luz, del material del que están formados los objetos y de la posición del observador. En este seminario se explica qué características tienen las fuentes de luz en OpenGL, cómo ajustar las propiedades del material y cuál es el modelo matemático que se usa para calcular el color en la superficie de un objeto.

Modelo de iluminación

Un modelo de iluminación es una función que, para cada longitud de onda, nos da la cantidad de luz (luminancia) que nos llega al punto de vista desde un punto de la escena. La luminancia depende de muchos factores como la forma, posición e intensidad de las fuentes de luz, la posición y características de los objetos a nuestro alrededor, la presencia de partículas suspendidas en el ambiente, etc.

Los modelos de iluminación "prácticos" hacen muchas simplificaciones para que se puedan calcular en tiempo real. La primera simplificación importante es la reducción del muestreo de la función a tres longitudes de onda primarias: el rojo, el verde y el azul. Así, el modelo de iluminación en un punto, idealmente continuo en la longitud de onda, se convierte en un vector de tres componentes, RGB, que hoy por hoy es el que nos sirve para asignar el color a los píxeles de nuestra imagen resultante. En OpenGL, cualquier característica que tenga que ver con magnitudes lumínicas se da como una tripleta RGB.

El modelo de iluminación que se utiliza en OpenGL es el siguiente:

$$C = E_m + A * A_m + \sum_i [A_{li} * A_m + D_{li} * D_m * \max(0, \boldsymbol{L}_i \cdot \boldsymbol{N}) + S_{li} * S_m * \max(0, \boldsymbol{H}_i \cdot \boldsymbol{N})^s]$$

donde el sufijo *m* se refiere a material y el sufijo *li* a la luz i-ésima. El significado de cada factor es el siguiente:

- C: Componente de color que se está calculando. Puede ser R, G o B. En el cálculo de C participará la misma componente de los parámetros de la función que sean colores.
- A: Color ambiental general independiente de las luces.
- E_m: Color emisivo del material.
- A_m: Color ambiental del material.
- D_m: Color difuso del material.
- S_m: Color especular del material.
- s: Exponente de concentración del brillo.
- A_{li}: Color ambiental de la luz i.
- D_{li}: Color difuso de la luz i.
- S_{li}: Color especular de la luz i.
- $\boldsymbol{L}_i$: Vector de iluminación de la luz i.
- $\boldsymbol{H}_i$: Vector intermedio para la luz i.
- $\boldsymbol{N}$: Vector normal en el punto de la superficie.

Así pues, para cada punto, el modelo se calcula tres veces, una para cada componente RGB. Pero, ¿qué valores toma cada componente de la tripleta RGB?, ¿depende de cada dispositivo o tarjeta gráfica? La solución más sencilla a este problema es tratar cada componente como un número real entre 0 y 1, y dejar que OpenGL se encargue luego de truncar *C* (si se pasa de 1, pues no debería ser menor que 0) y hacer la correspondencia con los colores verdaderos que puede tener cada pixel de la pantalla.

Por defecto no está activada la iluminación en OpenGL. Para activar la iluminación usaremos la siguiente orden:

```
glEnable(GL_LIGHTING)
```

A partir de ese momento todo lo que dibujemos saldrá iluminado según los ajustes que hayamos hecho de los parámetros del modelo de iluminación. Para desactivar la iluminación usaremos `glDisable` con el mismo parámetro.

Tipos y ajustes de fuentes de luz

La primera luz que aparece en el modelo de iluminación es una luz ambiental presente en todo punto que hemos llamado *A*. Si se quiere que el color RGB de *A* sea, por ejemplo, (0.3,0.2,0.1) usaremos las siguientes instrucciones:

```
GLfloat A[]={0.3, 0.2, 0.1, 1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, A);
```

Observamos que *A* tiene cuatro componentes. La última se usa para transparencias, que no nos concierne ahora, y aquí la ponemos a 1 sin más. Por defecto la luz ambiental general tiene el color (0.2,0.2,0.2,1.0) y está activa.

Aparte de la luz ambiente general podemos construir nuestras propias luces. ¿Cuántas? OpenGL permite como mínimo 8 luces diferentes (más que suficientes) que se nombran como `GL_LIGHT0`, `GL_LIGHT1`, ..., `GL_LIGHT7`. Es decir, nuestro sumatorio del modelo de iluminación tendrá un término por cada luz activa. Para activar la luz *i* simplemente hay que habilitarla:

```
glEnable(GL_LIGHTi)
```

Sólo las luces activas intervienen en la iluminación de la escena. Las luces activas se deshabilitan con `glDisable`.

Para ajustar los colores de los parámetros A_{l0}, D_{l0} y S_{l0} de la luz `GL_LIGHT0` en el modelo de iluminación, antes o después de activarla da lo mismo, lo haremos así:

```
GLfloat Al0[]={0.2,0.3,0.2,1.0};
GLfloat Dl0[]={0.5,0.7,1.0,1.0};
GLfloat Sl0[]={1.0,1.0,0.9,1.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, Al0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, Dl0);
glLightfv(GL_LIGHT0, GL_SPECULAR, Sl0);
```

Los colores escogidos son sólo un ejemplo. Para el resto de luces que se quieran activar se hace lo mismo variando el nombre de la luz.

Ejercicio S7E01: Dibujar una esfera de radio 0.5 vista desde (1.0,0.0,1.0) que esté iluminada con una luz amarilla. ¿De dónde viene la luz? ¿Y si variamos la posición de la cámara?

De los parámetros que dependen de la fuente de luz quedan por fijar los vectores ***L*** y ***H***. Ambos se derivan de la posición en la que se encuentra la fuente de luz. Por defecto la posición de la luz está en el punto (0,0,∞) y el vector ***L*** es el (0,0,1). Dependiendo de si la luz se encuentra en el infinito o en un punto del espacio distinguimos dos tipos de fuentes.

Fuentes de luz direccionales

La luz direccional está producida por una fuente que se encuentra suficientemente alejada como para considerarla en el infinito -como el Sol p.e.-. De una luz direccional podemos decidir su procedencia, esto es, el vector ***L***. Por ejemplo, si queremos que la luz `GL_LIGHT1` proceda de un punto alejado en la dirección del eje ***Y*** -iluminación cenital- fijaremos la posición de la luz así:

```
GLfloat posicion[]={0.0, 1.0, 0.0, 0.0};
glLightfv(GL_LIGHT1, GL_POSITION, posicion);
```

Es decisivo que la cuarta coordenada de la posición de la luz sea 0. Esto es lo que le indica a OpenGL que es direccional.

Fuentes de luz puntuales

Para definir una luz puntual, esto es, situada en un punto más o menos cercano a la escena, usaremos la misma orden, esta vez cuidando de poner la cuarta coordenada a 1:

```
GLfloat posicion[]={0.0, 1.0, 0.0, 1.0};
glLightfv(GL_LIGHT1, GL_POSITION, posicion);
```

Ejercicio S7E02: Comparar la iluminación de una esfera de radio 0.5 vista desde el punto (1,0,1) entre una luz amarilla direccional en (0,1,0,0) y otra puntual en (0,1,0,1).

La intensidad (color) de una luz puntual es, por defecto, la misma en todas las direcciones. Este comportamiento se conoce como emisión uniforme. Sin embargo, si observamos la iluminación que produce un flexo en la mesa o una farola de la carretera, distinguimos un cono de luz con mayor intensidad en el centro que en los costados. A este comportamiento se le conoce como emisión focalizada - en general no uniforme-.

Luces focales

Las luces focales son luces puntuales con emisión no uniforme. Forman un foco de luz donde la intensidad depende del ángulo de la dirección de iluminación respecto a la dirección central del foco. OpenGL nos permite formar luces puntuales focalizadas sin más que establecer:

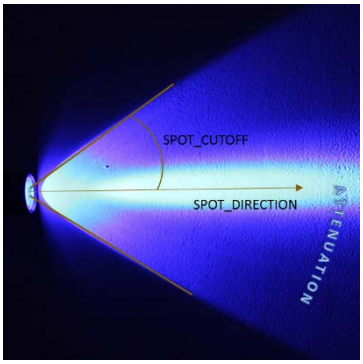


Fig. 1. Luz focal

1. La dirección central del foco de luz. Se da como el vector que parte de la luz y sigue la dirección principal de iluminación. Por ejemplo, si queremos que el foco apunte hacia abajo el vector sería el (0,-1,0). Para ello se usa la orden:

   ```
   GLfloat dir_central[]={0.0, -1.0, 0.0};
   glLightfv(GL_LIGHT2,GL_SPOT_DIRECTION,dir_central);
   ```

2. El ángulo de apertura del cono de luz. Se suele dar como el ángulo entre la dirección principal de iluminación y la dirección límite de iluminación a partir de la cual ya no ilumina el foco -de hecho es el semiángulo de apertura-.

   ```
   glLightf(GL_LIGHT2, GL_SPOT_CUTOFF, 30.0);
   ```

que corresponde a un foco que tiene un cono de 60 grados de apertura.

3. El factor de atenuación. Con este factor indicamos cómo queremos que vaya bajando la intensidad conforme nos alejamos de la dirección principal del foco. Los valores más altos indican bajadas más bruscas. Se sigue la ley de la potencia del coseno. Por ejemplo, si el factor de atenuación es cero la intensidad de la luz es la misma siempre que estemos dentro del cono iluminado. Para fijar el factor de atenuación se hace así:

   ```
   glLightf(GL_LIGHT2, GL_SPOT_EXPONENT, 2.0);
   ```

Se ha elegido la luz nº 2 como se podría haber elegido otra cualquiera. Los valores numéricos son ejemplos sin más.

Ejercicio S7E03: Iluminar una esfera de lado 0.5 con una luz focal amarilla situada en (0,1,0) con dirección principal de iluminación hacia (0,-1,0), con un semiángulo de apertura de 20º y con un factor de atenuación de 5. Situar la cámara en (1,1,1). Comparar los resultados con los ejercicios anteriores.

Sistema de coordenadas

Cabe preguntarse, cuándo se da la posición de una luz y en qué sistema de coordenadas estamos trabajando. La respuesta es sencilla pero las consecuencias hay que meditarlas. La posición de la luz sufre las mismas transformaciones que un vértice mandado a dibujar en ese momento. Esto quiere decir que, en el momento en que se lanza el `GL_POSITION`, se aplican a las coordenadas de la luz la matriz compuesta *modelview*. Por tanto, si queremos que **la luz sea independiente del movimiento de la cámara debemos indicar su posición después de fijar la cámara** como si de un vértice más se tratara.

Ejercicio S7E04: Construir una cámara con una luz focal adherida a ella (dependiente de la cámara).

Ajustes del material

Los últimos parámetros de nuestro modelo que quedan por fijar son los que dependen del material. El material lo entendemos aquí como la pintura con la hemos pintado la superficie del objeto y que nos da un color base y más o menos reflejos según sea mate o brillante. Del material podemos decidir su color emisivo E_m (como si emitiera luz propia), su color ambiental A_m (cuando no está iluminado), su color difuso D_m, el color en los brillos S_m y el exponente de concentración del brillo s.

Hay dos maneras de hacerlo. La primera es similar a la usada para las luces y se procede así:

```
GLfloat Em[]={0.1,0.1,0.2,1.0};
GLfloat Am[]={0.2,0.2,0.2,1.0};
GLfloat Dm[]={0.5,0.7,0.6,1.0};
GLfloat Sm[]={0.8,0.8,0.8,1.0};
GLfloat s=20.0;
glMaterialfv(GL_FRONT, GL_EMISSION, Em);
glMaterialfv(GL_FRONT,GL_AMBIENT,Am);
glMaterialfv(GL_FRONT,GL_DIFFUSE,Dm);
glMaterialfv(GL_FRONT,GL_SPECULAR,Sm);
glMaterialf(GL_FRONT, GL_SHININESS, s);
```

El primer parámetro indica la cara del polígono afectada, si es la frontal `GL_FRONT`, si la trasera `GL_BACK` o si ambas `GL_FRONT_AND_BACK`. El segundo parámetro identifica la característica que se quiere fijar del material y el tercero su nuevo valor.

La manera anterior deshabilita la influencia de `glColor()` sobre el color de los vértices. Si se quiere preservar la validez de `glColor` puede procederse de esta otra manera:

```
glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);
….
glColor3f(0.5,0.7,0.6);
```

De esta manera indicamos qué característica queremos enganchar a `glColor()`. En el ejemplo anterior el color difuso del material cambia en cuanto se procesa la orden `glColor3f(…)`. Es importante habilitar este modo de proceder con `glEnable` pues es sólo a partir de ese momento que tiene efecto la asociación. Naturalmente la asociación solo afecta a aquellas características que son colores y para la cara del polígono que se especifica.

Ejercicio S7E05: Construir una matriz de 16 esferas (4x4) donde en las filas se varíe el color difuso y la concentración de brillo y en las columnas el color especular.

Normales

Una característica del objeto relacionada con su geometría (no con su apariencia como las anteriores) es el vector $\boldsymbol{N}$ normal a la superficie en cada punto. Como hemos visto al principio del seminario este vector es muy importante en el modelo de iluminación. Tanto es así que si queremos tener iluminación es necesario indicar la normal con la que queremos que se calcula la iluminación con la orden:

```
glNormal3f(nx,ny,nz)
```

donde *nx*,*ny*,*nz* son las componentes del vector perpendicular a la superficie. Es necesario que este vector sea unitario y se conserve unitario aunque se hagan transformaciones (escalados por ejemplo). Si las normales no se mantienen unitarias se ve enseguida que los brillos no salen bien pues saturan si nos alejamos o desaparecen cuando estamos cerca. Si no queremos preocuparnos del tema vale la pena usar la orden:

```
glEnable(GL_NORMALIZE)
```

Esto hace que las normales siempre se hagan unitarias antes de calcular la iluminación.

Modelo de sombreado

Cabe preguntarse ahora ¿dónde se calcula el modelo de iluminación y cómo se colorean los pixeles? Supongamos que tenemos un polígono definido por sus vértices. La forma de visualizar el polígono en pantalla es calcular qué pixeles corresponden a su interior y colorearlos. El proceso de cálculo de los pixeles se conoce como proceso de rasterización y consiste en muestrear el espacio continuo de la superficie del polígono en R^3 para formar una colección de pixeles en N^2. Lo lógico sería que para cada píxel se aplicara el modelo de iluminación a su correspondiente punto en R^3. Sin embargo este proceso es demasiado lento (en tubería fija) y se opta por una de estas dos opciones:

a. **Modelo de sombreado constante**. Poner todos los pixeles al color indicado por el modelo de iluminación calculado en uno de los vértices del polígono -el último-.
b. **Modelo de sombreado suave o de Gouraud**. Calcular el modelo de iluminación en cada vértice del polígono y hacer una interpolación (bilineal) para calcular el color de cada píxel.

El cálculo del modelo de iluminación en un vértice no tiene mayor problema pues sólo hay que echar mano de la máquina de estados de OpenGL para conocer cuál es la normal actual (el último `glNormal` dado), el material actual (los valores últimos fijados con `glMaterial`) y las características de las luces

activas. Por supuesto todo este cálculo, incluida la interpolación del color entre pixeles, lo hace OpenGL (en tubería fija).

Para elegir el modelo de sombreado que se quiere aplicar hay que usar la orden:

`glShadeModel(tipo)`

donde `tipo` puede ser `GL_FLAT` para sombreado constante o `GL_SMOOTH` para sombreado de Gouraud.

Ejercicio S7E06: Dibujar dos teteras iluminadas desde arriba con diferentes modelos de sombreado una al lado de la otra.

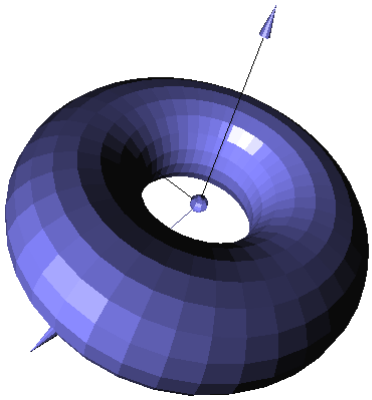


Figura 2. Sombreado constante

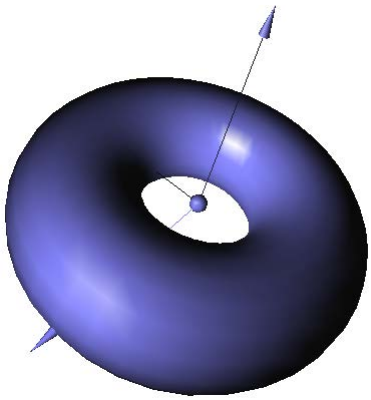


Figura 3. Sombreado de Gouraud

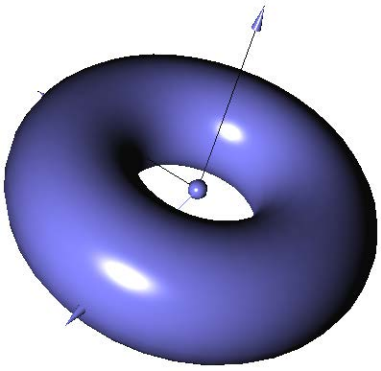


Figura 4. Sombreado de Phong (no posible en tubería fija)

Valores por defecto

Generales

(A) GL_LIGHT_MODEL_AMBIENT	{0.2, 0.2, 0.2, 1.0}
GL_SHADE_MODEL	GL_SMOOTH

Luces

(A_{li}) GL_AMBIENT	{0.0, 0.0, 0.0, 1.0}
(D_{li}) GL_DIFFUSE	LIGHT0 {1.0, 1.0, 1.0, 1.0}, resto {0.0, 0.0, 0.0, 1.0}
(S_{li}) GL_SPECULAR	LIGHT0 {1.0, 1.0, 1.0, 1.0}, resto {0.0, 0.0, 0.0, 1.0}
GL_POSITION	{0.0, 0.0, 1.0, 0.0}
GL_SPOT_DIRECTION	{0.0, 0.0, -1.0}
GL_SPOT_EXPONENT	0.0
GL_SPOT_CUTOFF	180º

Material

(A_m) GL_AMBIENT	{0.2, 0.2, 0.2, 1.0}
(D_m) GL_DIFFUSE	{0.8, 0.8, 0.8, 1.0}
(S_m) GL_SPECULAR	{0.0, 0.0, 0.0, 1.0}
(E_m) GL_EMISSION	{0.0, 0.0, 0.0, 1.0}
(s) GL_SHININESS	0.0

RESUMEN

Modelo de iluminación en OpenGL

$$C = E_m + A * A_m + \sum_i [A_{li} * A_m + D_{li} * D_m * \max(0, \boldsymbol{L}_i \cdot \boldsymbol{N}) + S_{li} * S_m * \max(0, \boldsymbol{H}_i \cdot \boldsymbol{N})^s]$$

	Parámetro	Descripción	Valor por defecto
Generales	A	Color ambiente genérico independiente de las luces	`0.2, 0.2, 0.2`
Dependientes del material	E_m	Color emisivo del material	`0.0, 0.0, 0.0`
	A_m	Color ambiental del material	`0.2, 0.2, 0.2`
	D_m	Color difuso del material	`0.8, 0.8, 0.8`
	S_m	Color especular del material	`0.0, 0.0, 0.0`
	s	Concentración del brillo del material	`0`
Dependientes de las fuentes de luz	A_{li}	Color ambiental de la luz	`0.0, 0.0, 0.0`
	D_{li}	Color difuso de la luz	`LIGHT0 1.0, 1.0, 1.0` `Resto 0.0, 0.0, 0.0`
	S_{li}	Color especular de la luz	`LIGHT0 1.0, 1.0, 1.0` `Resto 0.0, 0.0, 0.0`
Dependientes de la posición y la forma	$\boldsymbol{L}_i$	Vector de iluminación	`0.0, 0.0, 1.0`
	$\boldsymbol{H}_i$	Vector intermedio entre el de iluminación y el de observación	`0.0, 0.0, 1.0`
	$\boldsymbol{N}$	Normal en el punto	`0.0, 0.0, 1.0`

Modelo de sombreado en OpenGL

Modelo plano: Por polígono, se usa la normal del último vértice.

Modelo suave: Por vértice, cada vértice aporta su normal. Es el de defecto.

Ejercicio S7E01: Dibujar una esfera de radio 0.5 vista desde (1.0,0.0,1.0) que esté iluminada con una luz amarilla. ¿De dónde viene la luz? ¿Y si variamos la posición de la cámara?

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

Dibujo de una esfera iluminada

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S7E01::Esfera iluminada"

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

void init()
// Funcion de inicializacion propia
{
        // Mensajes por consola
        std::cout << PROYECTO << " running" << std::endl;

        glClearColor(0,0,0,1);                  // Color de fondo

        // Inicialización de luces
        GLfloat Al0[]={0.2,0.2,0.2,1.0};        // Color ambiental de la luz
        GLfloat Dl0[]={1.0,1.0,0.0,1.0};        // Color difuso de la luz
        GLfloat Sl0[]={1.0,1.0,0.0,1.0};        // Color especular de la luz
        glLightfv(GL_LIGHT0, GL_AMBIENT, Al0);  // Caracteristicas de LIGHT0
        glLightfv(GL_LIGHT0, GL_DIFFUSE, Dl0);
        glLightfv(GL_LIGHT0, GL_SPECULAR, Sl0);
        glEnable(GL_LIGHT0);

        // Caracteristicas del render
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING);
}

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();

  /* Aunque variemos la posición de la cámara, la esfera siempre se ve iluminada de frente
     como si la luz estuviera donde esta el observador */

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

  glColor3f(1,1,1);
  glutSolidSphere(0.5,20,20);                            // Dibuja una esfera blanca

  glutSwapBuffers();                                     // 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(90,razon,0.1,10);
}
```

```
void main(int argc, char** argv)
// Programa principal
{
  glutInit(&argc, argv);                                  // Inicializacion de GLUT
  glutInitDisplayMode(GLUT_DOUBLE|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
  glutDisplayFunc(display);                               // Alta de la funcion de atencion a display
  glutReshapeFunc(reshape);                               // Alta de la funcion de atencion a reshape
  init();                                                 // Funcion propia de inicializacion
  glutMainLoop();                                         // Puesta en marcha del programa
}
```

Ejercicio S7E02: Comparar la iluminación de una esfera de radio 0.5 vista desde el punto (1,0,1) entre una luz amarilla direccional en (0,1,0,0) y otra puntual en (0,1,0,1).

```
/*************************************************
ISGI::Luz Puntual vs Direccional
Roberto Vivo', 2013 (v1.0)

Dibujo de una esfera iluminada con diferentes luces

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S7E02::Puntual vs Direccional"

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

static bool DIRECCIONAL(true);                  // Tipo de luz

void init()
// Funcion de inicializacion propia
{
        // Mensajes por consola
        std::cout << PROYECTO << " running" << std::endl;
        std::cout << "d: Luz direccional" << std::endl;
        std::cout << "p: Luz puntual" << std::endl << std::endl;

        glClearColor(0,0,0,1);                  // Color de fondo

        // Inicialización de luces
        GLfloat Al0[]={0.2,0.2,0.2,1.0};        // Color ambiental de la luz
        GLfloat Dl0[]={1.0,1.0,0.0,1.0};        // Color difuso de la luz
        GLfloat Sl0[]={1.0,1.0,0.0,1.0};        // Color especular de la luz
        glLightfv(GL_LIGHT0, GL_AMBIENT, Al0);  // Caracteristicas de LIGHT0
        glLightfv(GL_LIGHT0, GL_DIFFUSE, Dl0);
        glLightfv(GL_LIGHT0, GL_SPECULAR, Sl0);
        glEnable(GL_LIGHT0);

        // Caracteristicas del render
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING);
}

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

  // Posiciona y define el tipo de luz
  GLfloat posicion[] = {0,1,0,0};
  if(DIRECCIONAL) posicion[3] = 0;                      // Luz direccional
  else posicion[3] = 1;                                 // Luz puntual
  glLightfv(GL_LIGHT0,GL_POSITION,posicion);
  glColor3f(1,1,1);
  glutSolidSphere(0.5,20,20);                           // Dibuja una esfera blanca

  glutSwapBuffers();                                    // 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(90,razon,0.1,10);
}

void onKey(unsigned char tecla, int x, int y)
// Funcion de atencion al teclado
{
        switch(tecla){
        case 'p':
                DIRECCIONAL = false;
                std::cout<<"PUNTUAL ON, direccional off"<<std::endl;
                break;
        case 'd':
                DIRECCIONAL = true;
                std::cout<<"DIRECCIONAL ON, puntual off"<<std::endl;
                break;
        case 27:                                                // Pulso escape
                exit(0);
        }
        glutPostRedisplay();
}

void main(int argc, char** argv)
// Programa principal
{
  glutInit(&argc, argv);                                // Inicializacion de GLUT
  glutInitDisplayMode(GLUT_DOUBLE|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
  glutDisplayFunc(display);                             // Alta de la funcion de atencion a display
  glutReshapeFunc(reshape);                             // Alta de la funcion de atencion a reshape
  glutKeyboardFunc(onKey);                              // Alta de funcion de atencion al teclado
  init();                                               // Funcion propia de inicializacion
  glutMainLoop();                                       // Puesta en marcha del programa
}
```

Ejercicio S7E03: Iluminar una esfera de lado 0.5 con una luz focal amarilla situada en (0,1,0) con dirección principal de iluminación hacia (0,-1,0), con un semiángulo de apertura de 20º y con un factor de atenuación de 5. Situar la cámara en (1,1,1). Comparar los resultados con los ejercicios anteriores.

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

Dibujo de una esfera iluminada con diferentes luces

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S7E03::Luz Focal"

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

void init()
// Funcion de inicializacion propia
{
        // Mensajes por consola
        std::cout << PROYECTO << " running" << std::endl;
        std::cout << "d: Luz direccional" << std::endl;
        std::cout << "p: Luz puntual" << std::endl;
        std::cout << "f: Luz focal" << std::endl << std::endl;

        glClearColor(0,0,0,1);                                  // Color de fondo

        // Inicialización de luces
        // LIGHT0: DIRECCIONAL
        GLfloat Al0[]={0.2,0.2,0.2,1.0};                        // Color ambiental de la luz
        GLfloat Dl0[]={1.0,1.0,0.0,1.0};                        // Color difuso de la luz

        GLfloat Sl0[]={1.0,1.0,0.0,1.0};                        // Color especular de la luz
        glLightfv(GL_LIGHT0, GL_AMBIENT, Al0);                  // Caracteristicas de LIGHT0
        glLightfv(GL_LIGHT0, GL_DIFFUSE, Dl0);
        glLightfv(GL_LIGHT0, GL_SPECULAR, Sl0);

        // LIGHT1: PUNTUAL
        GLfloat Al1[]={0.2,0.2,0.2,1.0};                        // Color ambiental de la luz
        GLfloat Dl1[]={1.0,1.0,0.0,1.0};                        // Color difuso de la luz

        GLfloat Sl1[]={1.0,1.0,0.0,1.0};                        // Color especular de la luz
        glLightfv(GL_LIGHT1, GL_AMBIENT, Al1);                  // Caracteristicas de LIGHT1
        glLightfv(GL_LIGHT1, GL_DIFFUSE, Dl1);
        glLightfv(GL_LIGHT1, GL_SPECULAR, Sl1);

        // LIGHT2: FOCAL
        GLfloat Al2[]={0.2,0.2,0.2,1.0};                        // Color ambiental de la luz
        GLfloat Dl2[]={1.0,1.0,0.0,1.0};                        // Color difuso de la luz
        GLfloat Sl2[]={1.0,1.0,0.0,1.0};                        // Color especular de la luz
        glLightfv(GL_LIGHT2, GL_AMBIENT, Al2);                  // Caracteristicas de LIGHT2
        glLightfv(GL_LIGHT2, GL_DIFFUSE, Dl2);
        glLightfv(GL_LIGHT2, GL_SPECULAR, Sl2);
        glLightf(GL_LIGHT2, GL_SPOT_CUTOFF, 20.0);
        glLightf(GL_LIGHT2, GL_SPOT_EXPONENT, 5.0);
        glEnable(GL_LIGHT2);

        // Caracteristicas del render
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING);
}

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

  // Posiciona y define el tipo de luz
  GLfloat posicion[] = {0,1,0,0};
  glLightfv(GL_LIGHT0,GL_POSITION,posicion);             // Luz direccional

  posicion[3] = 1;
  glLightfv(GL_LIGHT1,GL_POSITION,posicion);             // Luz puntual

  GLfloat dir_central[]={0.0, -1.0, 0.0};
  glLightfv(GL_LIGHT2, GL_SPOT_DIRECTION, dir_central);
  glLightfv(GL_LIGHT2,GL_POSITION,posicion);             // Luz focal

  glColor3f(1,1,1);
  glutSolidSphere(0.5,200,200);                          // Dibuja una esfera blanca

  glutSwapBuffers();                                     // 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(90,razon,0.1,10);
}

void onKey(unsigned char tecla, int x, int y)
// Funcion de atencion al teclado
{
        switch(tecla){
        case 'p':
                glDisable(GL_LIGHT0);
                glDisable(GL_LIGHT2);
                glEnable(GL_LIGHT1);
                std::cout<<"LUZ PUNTUAL"<<std::endl;
                break;
        case 'd':
                glDisable(GL_LIGHT1);
                glDisable(GL_LIGHT2);
                glEnable(GL_LIGHT0);
                std::cout<<"LUZ DIRECCIONAL"<<std::endl;
                break;
        case 'f':
                glDisable(GL_LIGHT0);
                glDisable(GL_LIGHT1);
                glEnable(GL_LIGHT2);
                std::cout<<"LUZ FOCAL"<<std::endl;
                break;
        case 27:                                                // Pulso escape
                exit(0);
        }
        glutPostRedisplay();
}

void main(int argc, char** argv)
// Programa principal
{
  glutInit(&argc, argv);                                 // Inicializacion de GLUT
  glutInitDisplayMode(GLUT_DOUBLE|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
  glutDisplayFunc(display);                              // Alta de la funcion de atencion a display
  glutReshapeFunc(reshape);                              // Alta de la funcion de atencion a reshape
  glutKeyboardFunc(onKey);                               // Alta de funcion de atencion al teclado
  init();                                                // Funcion propia de inicializacion
  glutMainLoop();                                        // Puesta en marcha del programa
}
```

Ejercicio S7E04: Construir una cámara con una luz focal adherida a ella (dependiente de la cámara).

```
/*************************************************
ISGI::Luz de Minero
Roberto Vivo', 2013 (v1.0)

Mueve la camara en la interfaz de Bola de Cristal
llevando una luz focal adherida

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S7E04::Luz de Minero"

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

// Variables globales
static int xantes,yantes;                       // Valor del pixel anterior
static float girox=0,giroy=0;                   // Valor del giro a acumular
static float escalado=1;                        // Valor del escalado acumulado
static float coef[16];                          // Matriz MODELVIEW
static enum Interaccion {GIRO,ESCALADO} accion;  // Tipo de acción de inspección

void init()
// Funcion propia de inicializacion
{
        cout << PROYECTO << " running" << endl;
        cout << "Version: OpenGL " <<  glGetString(GL_VERSION) << endl;
        cout << "Arrastre con boton izquierdo: Mueve la camara en la Bola" << endl;
        cout << "Arrastre con boton derecho: Acerca o aleja la camara" << endl;
        cout << "q: Posicion inicial de la camara" << endl;
        cout << "esc: Salir" << endl;

        glClearColor(0.0,0.0,0.0,1.0);                  // Color de fondo a negro

        glGetFloatv(GL_MODELVIEW_MATRIX, coef);         // Inicializa coef a la matriz identidad

        // Inicializa luces
        // LIGHT2: FOCAL
        GLfloat Al2[]={0.2,0.2,0.2,1.0};                // Color ambiental de la luz
        GLfloat Dl2[]={1.0,1.0,0.0,1.0};                // Color difuso de la luz
        GLfloat Sl2[]={1.0,1.0,0.0,1.0};                // Color especular de la luz
        glLightfv(GL_LIGHT2, GL_AMBIENT, Al2);          // Caracteristicas de LIGHT2
        glLightfv(GL_LIGHT2, GL_DIFFUSE, Dl2);
        glLightfv(GL_LIGHT2, GL_SPECULAR, Sl2);
        glLightf(GL_LIGHT2, GL_SPOT_CUTOFF, 10.0);
        glLightf(GL_LIGHT2, GL_SPOT_EXPONENT, 5.0);
        glEnable(GL_LIGHT2);

        // Caracteristicas del render
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING);}

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

  // Para la luz de minero situamos la luz antes de situar la camara
  // asi no le afecta la VIEW y trabajamos en coordenadas del sistema de la camara
  GLfloat posicion[] = {0,0,0,1};
  glLightfv(GL_LIGHT2,GL_POSITION,posicion);            // Luz focal sobre la camara
  GLfloat dir_central[]={0.0, 0.0, -1.0};               // Direccion del foco la de la vista
  glLightfv(GL_LIGHT2, GL_SPOT_DIRECTION, dir_central);

  // Los giros se acumulan con signo contrario por la izquierda sobre el objeto girado,
  // así siempre se gira sobre el sistema fijo. VIEW*R*Racumulado*zoom
```

```
  // Primero calculamos los giros R*Racumulado (signos contrarios)
  glRotatef(-girox,1,0,0);                              // 3.Aplicamos el giro actual en X fijo
  glRotatef(-giroy,0,1,0);                              // 2.Aplicamos el giro actual en Y fijo
  glMultMatrixf(coef);                                  // 1.Aplicamos el giro anterior
  glGetFloatv(GL_MODELVIEW_MATRIX, coef);               // Nos guardamos la orientacion

  // Restablecemos la MODELVIEW y componemos en el orden correcto
  glLoadIdentity();
  gluLookAt(0,0,5,0,0,0,0,1,0);                         // Situa la camara fija MV=VIEW
  glMultMatrixf(coef);                                  // Gira el acumulado MV=VIEW*R*Racum
  glScalef(escalado,escalado,escalado);                 // Escala el modelo MV=VIEW*R*Racum*zoom

  // Dibuja la escena
  glutSolidTeapot(1.0);

  glutSwapBuffers();                                    // Intercambia los buffers
}

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 PERSPECTIVA */
        gluPerspective(60,razon,1,10);
}

void onClick(int button,int state, int x, int y)
// Funcion de atencion al boton del raton
// button: GLUT_LEFT|MIDDLE|RIGHT_BUTTON
// state: GLUT_UP|DOWN
// x,y: pixel respecto a vertice superior izquierdo
{
        //La inspección puede ser girando la pieza o variando el tamaño
        switch (button)
        {
                case GLUT_LEFT_BUTTON:                  // Girar la pieza
                        accion= GIRO;
                        xantes=x;                       // Guarda el valor del pixel picado
                        yantes=y;
                        break;
                case GLUT_RIGHT_BUTTON:                 // Escalar la pieza
                        accion= ESCALADO;
                        yantes=y;               // La escala se maneja con mvto. vertical del ratón
                        girox=0;                // Cuando se escala no se acumula ningún giro
                        giroy=0;
                        break;
        };
}

void onMotion(int x, int y)
// Funcion de atencion al raton con el boton pulsado
// x,y: coordenadas del cursor referidas al pixel superior izquierdo(0,0)
{
        static const float pix2deg = 1.0;               // Factor de conversión pixel a grados
        static const float pix2fac = 0.01;              // Factor de conversión pixel a escalado

        switch(accion){                                 // La accion la determina el boton pulsado
        case GIRO:                              // La acumulación del giro se produce en la MODELVIEW
                girox= (y - yantes) * pix2deg;  // y crece hacia abajo. giro antihorario en x
                giroy= (x - xantes) * pix2deg;  // x crece hacia derecha. giro antihorario en y
                yantes=y;
                xantes=x;
        break;
        case ESCALADO:                          // La acumulación del escalado se lleva en "escalado"
                escalado+= (yantes - y) * pix2fac;// y crece hacia abajo. escalado crece hacia arriba
```

```
                yantes=y;
        break;
        };

        glutPostRedisplay();
}

void onKey(unsigned char tecla, int x, int y)
// Funcion de atencion al teclado
{
        switch(tecla){
        case 'q':                               // Vuelve a la posicion original
                girox = 0;
                giroy = 0;
                escalado = 1;
                glPushMatrix();
                glLoadIdentity();
                glGetFloatv(GL_MODELVIEW_MATRIX,coef);   // Inicializa coef a la identidad
                glPopMatrix();
                break;
        case 27:                                                // Salir de la aplicacion
                exit(0);
        }
        glutPostRedisplay();
}

void main(int argc, char** argv)
// Programa principal
{
  glutInit(&argc, argv);                                // Inicializacion de GLUT
  glutInitDisplayMode(GLUT_DOUBLE|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
  glutDisplayFunc(display);             // Alta de la funcion de atencion a display
  glutReshapeFunc(reshape);             // Alta de la funcion de atencion a reshape
  glutMouseFunc(onClick);               // Alta de la funcion de atencion al click del ratón
  glutMotionFunc(onMotion);             // Alta de la funcion de atencion al movimiento del ratón
  glutKeyboardFunc(onKey);              // Alta de la funcion de atencion al teclado
  init();                               // Inicializacion propia
  glutMainLoop();                       // Puesta en marcha del programa
}
```

Ejercicio S7E05: Construir una matriz de 16 esferas (4x4) donde en las filas se varíe el color difuso y la concentración de brillo y en las columnas el color especular.

```
/*************************************************
ISGI::Variacion del material
Roberto Vivo', 2013 (v1.0)

Dibujo de 16 esferas con diferente material

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S7E05::Variacion del material"

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

void init()
// Funcion de inicializacion propia
{
        // Mensajes por consola
        std::cout << PROYECTO << " running" << std::endl;

        glClearColor(0,0,0,1);                  // Color de fondo

        // Inicialización de luces
        GLfloat Al0[]={0.2,0.2,0.2,1.0};        // Color ambiental de la luz
        GLfloat Dl0[]={1.0,1.0,1.0,1.0};        // Color difuso de la luz
        GLfloat Sl0[]={1.0,1.0,1.0,1.0};        // Color especular de la luz
        glLightfv(GL_LIGHT0, GL_AMBIENT, Al0);  // Caracteristicas de LIGHT0
        glLightfv(GL_LIGHT0, GL_DIFFUSE, Dl0);
        glLightfv(GL_LIGHT0, GL_SPECULAR, Sl0);
        glEnable(GL_LIGHT0);

        // Caracteristicas del render
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING);
}

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

  GLfloat posicion[] = {1,1,1,0};                        // Situa la luz direccional
  glLightfv(GL_LIGHT0, GL_POSITION, posicion);

  // Dibuja 16 esferas

  glTranslatef(-1.5,-1.5,0);
  for(int i=0; i<4; i++){
          GLfloat difuso[] = {0.2*i,0.2*i,0.1*i,1};
          glMaterialfv(GL_FRONT, GL_DIFFUSE, difuso);
          for(int j=0; j<4; j++){
              GLfloat especular[] = {0.3*j,0.3*j,0.3*j,1};
              glMaterialfv(GL_FRONT, GL_SPECULAR, especular);
                  glMaterialf(GL_FRONT, GL_SHININESS, 100);
                  glPushMatrix();
                  glTranslatef(i,j,0);
                  glutSolidSphere(0.5,200,200);          // Dibuja una esfera
                  glPopMatrix();
          }
  }

  glutSwapBuffers();                                     // 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;
        if(w<h)
                glOrtho(-2,2,-2/razon,2/razon, 0,10);
        else
                glOrtho(-2*razon,2*razon,-2,2, 0,10);
}

void main(int argc, char** argv)
// Programa principal
{
  glutInit(&argc, argv);                                        // Inicializacion de GLUT
  glutInitDisplayMode(GLUT_DOUBLE|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
  glutDisplayFunc(display);             // Alta de la funcion de atencion a display
  glutReshapeFunc(reshape);             // Alta de la funcion de atencion a reshape
  init();                               // Funcion propia de inicializacion
  glutMainLoop();                       // Puesta en marcha del programa
}
```

Ejercicio S7E06: Dibujar dos teteras iluminadas desde arriba con diferentes modelos de sombreado una al lado de la otra.

```
/*************************************************
ISGI::Modelo de Sombreado
Roberto Vivo', 2013 (v1.0)

Dibujo de 2 teteras con diferente modelo de sombreado

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S7E06::Modelo de sombreado"

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

void init()
// Funcion de inicializacion propia
{
        // Mensajes por consola
        std::cout << PROYECTO << " running" << std::endl;

        glClearColor(0,0,0,1);                          // Color de fondo

        // Inicialización de luces
        GLfloat Al0[]={0.2,0.2,0.2,1.0};                // Color ambiental de la luz
        GLfloat Dl0[]={1.0,1.0,1.0,1.0};                // Color difuso de la luz

        GLfloat Sl0[]={1.0,1.0,1.0,1.0};                // Color especular de la luz
        glLightfv(GL_LIGHT0, GL_AMBIENT, Al0);          // Caracteristicas de LIGHT0
        glLightfv(GL_LIGHT0, GL_DIFFUSE, Dl0);
        glLightfv(GL_LIGHT0, GL_SPECULAR, Sl0);
        glEnable(GL_LIGHT0);

        // Materiales
        GLfloat difuso[] = {0.8,0.7,0.2,1};
        glMaterialfv(GL_FRONT, GL_DIFFUSE, difuso);
        GLfloat especular[] = {0.6,0.6,0.5,1};
        glMaterialfv(GL_FRONT, GL_SPECULAR, especular);
        glMaterialf(GL_FRONT, GL_SHININESS, 100);


        // Caracteristicas del render
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING);
}

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

  GLfloat posicion[] = {1,1,1,0};                       // Situa la luz direccional
  glLightfv(GL_LIGHT0, GL_POSITION, posicion);

  // Dibuja dos teteras

  glShadeModel(GL_FLAT);                                // Modelo de iluminacion constante
  glPushMatrix();
  glTranslatef(-2,0,0);
  glutSolidTeapot(1.0);
  glPopMatrix();

  glShadeModel(GL_SMOOTH);                              // Modelo de iluminacion suave
  glPushMatrix();
  glTranslatef(2,0,0);
  glutSolidTeapot(1.0);
```

```
  glPopMatrix();


  glutSwapBuffers();                                    // 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;
        if(w<h)
                glOrtho(-4,4,-4/razon,4/razon, 0,10);
        else
                glOrtho(-4*razon,4*razon,-4,4, 0,10);
}

void main(int argc, char** argv)
// Programa principal
{
  glutInit(&argc, argv);                                            // Inicializacion de GLUT
  glutInitDisplayMode(GLUT_DOUBLE|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
  glutDisplayFunc(display);                         // Alta de la funcion de atencion a display
  glutReshapeFunc(reshape);                         // Alta de la funcion de atencion a reshape
  init();                                           // Funcion propia de inicializacion
  glutMainLoop();                                   // Puesta en marcha del programa
}
```