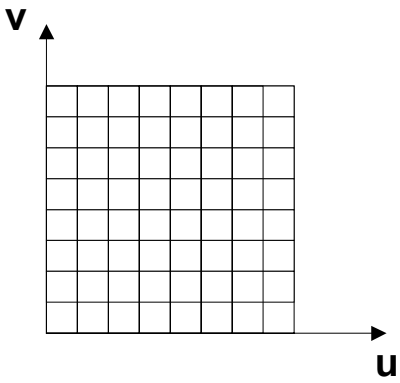
CAPITULO 7

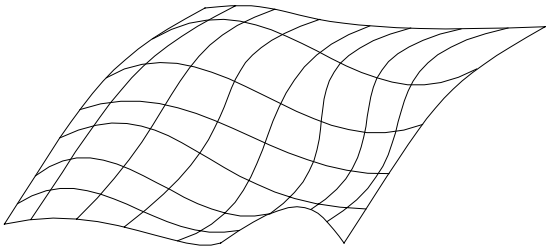
Textura

7.1 Mapeo (u,v)→(x,y,z)

La forma más simple de darle textura a una superficie es copiar los píxeles desde una textura plana a una superficie cualquiera.

Esto es:



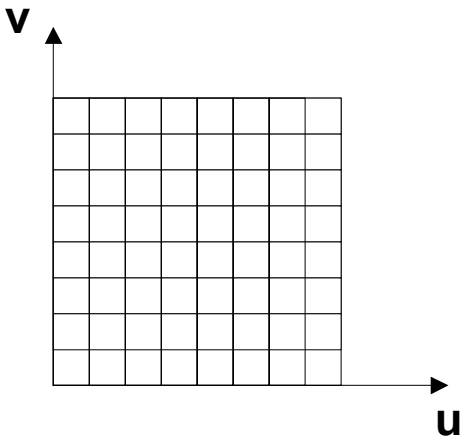


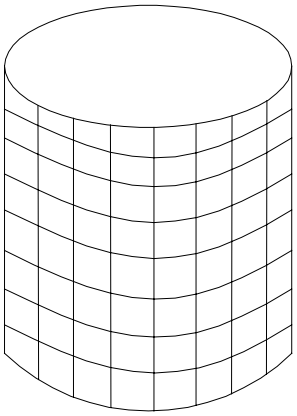
(x,y,z)

$(u,v) \in [0,1]$

Un primer procedimiento es parametrizar la superficie y donde entonces establecer una correspondencia con u,v.

Ejemplo: Caso Cilindro

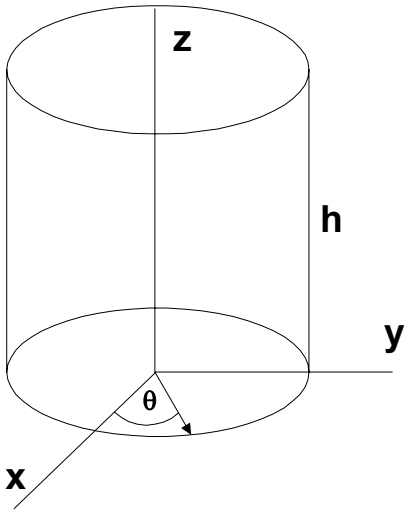




(x,y,z)

Para el caso del cilindro, tenemos que una forma paramétrica de sus ecuaciones está dada por:

$$x = r\cos\theta$$
$$y = rsen\theta$$
$$z = h$$



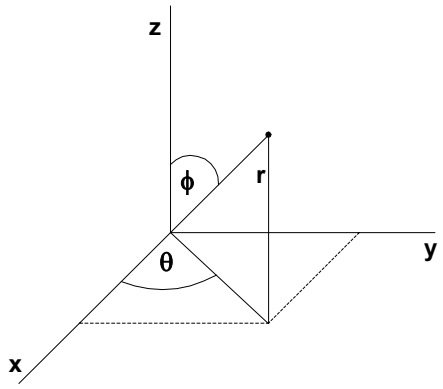
Donde *h* es la altura.

$$0 < \theta < 2\pi$$
$$0 < z < 1$$

Así podemos asociar puntos *u,v* con puntos dados por θ,z. Donde:

$$u = \frac{\theta}{2\pi}$$
$$v = z$$

Caso esfera



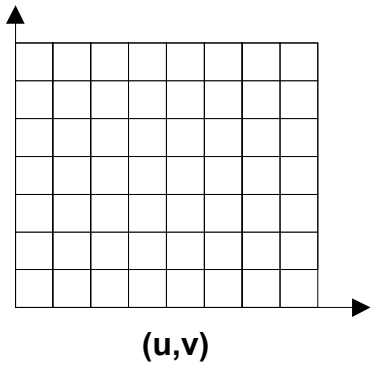
$$x = r\cos\theta\, sen\phi$$
$$y = rsen\theta\, sen\phi$$
$$z = r\cos\phi$$

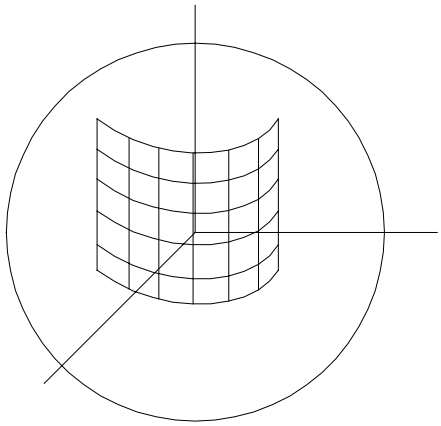
Luego tenemos que:

$$0 < \theta < 2\pi$$
$$0 < \phi < \frac{\pi}{2}$$

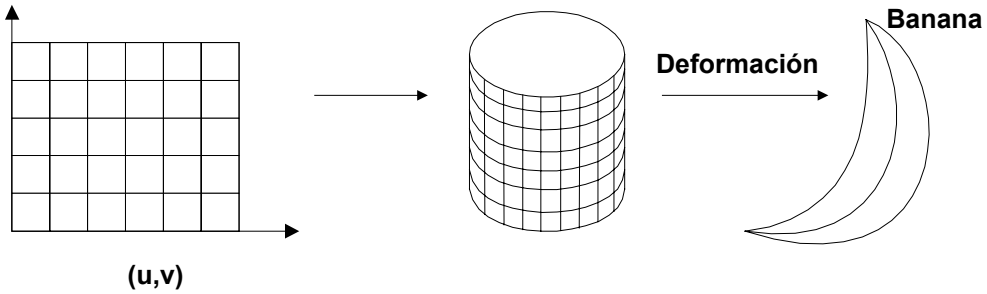
$$u = \frac{\theta}{2\pi}$$
$$v = \frac{2\phi}{\pi}$$

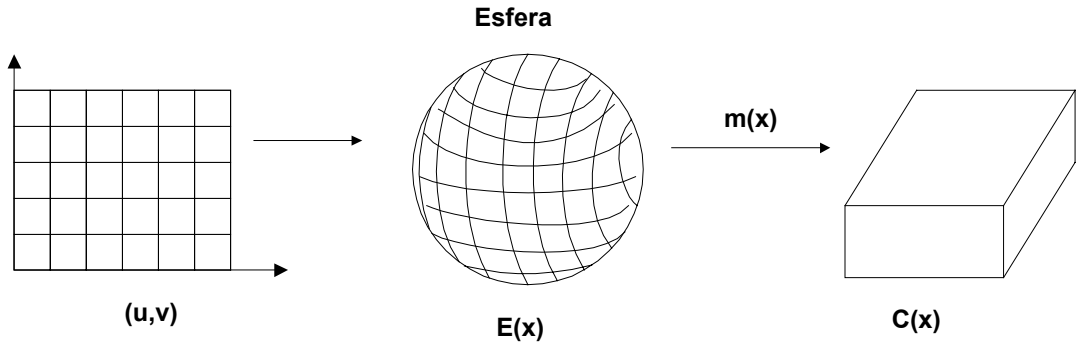
También se puede subdividir la esfera en hemisferios, cambiando los límites donde son válidos los parámetros θ y ϕ:





Este mapeo puede producir buen resultado y se puede usar la siguiente técnica, por ejemplo:



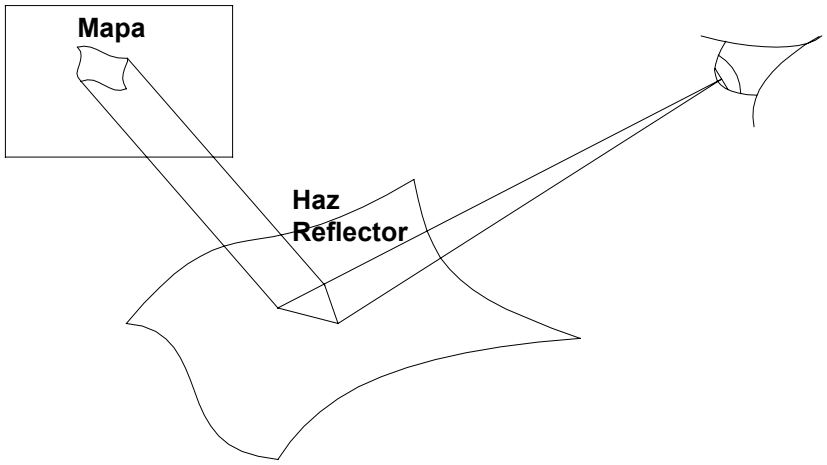


$$m(x) = \lambda E(x) - (1-\lambda) C(x) \qquad \lambda \in [0,1]$$

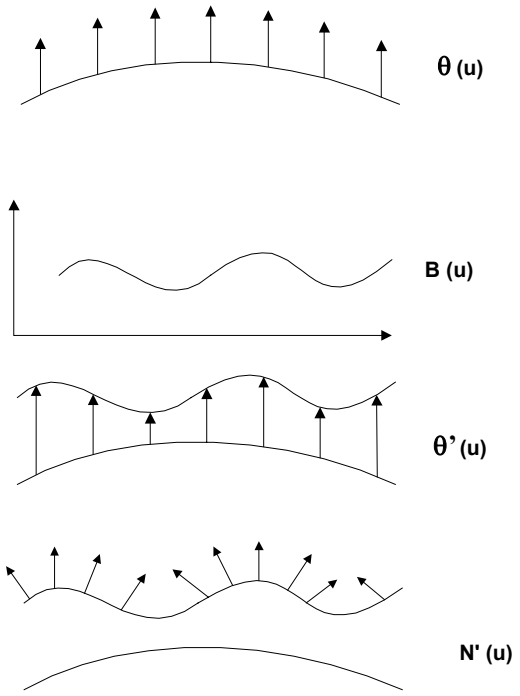
Por deformación continua.

7.2 Mapeo dependiente del observador

Una forma simple de realizar un mapeo es mediante la reflexión del plano *(u,v)* en la figura que se está desplegando. Por ejemplo:



7.2.1 Mapeo Bump



$$\theta'(u,v) = \theta(u,v) + B(u,v)\frac{N}{|N|}$$

$$\theta'_u(u,v) = \theta_u + B_u\frac{N}{|N|} + B\left[\frac{N}{|N|}\right]_u$$

$$\theta'_v(u,v) = \theta_v + B_v\frac{N}{|N|} + B\left[\frac{N}{|N|}\right]_v$$

$$N'(u,v) = \theta'_u(u,v) \times \theta'_v(u,v)$$

$$N'(u,v) = \theta_u \times \theta_v + B_u\left\{\frac{N}{|N|} \times \theta_v\right\} + B_v\left\{\theta_u \times \frac{N}{|N|}\right\} + B_u B_v\left\{\frac{N \times N}{|N|^2}\right\}$$

El último término es cero por el producto cruz. Luego:

$$N' = N + D$$

donde

$$D = B_u(N \times \theta_v) - B_v(N \times \theta_u)$$

N se entiende normalizado.

7.2.2 Sistema de Partículas

Corresponde a la categoría de un sistema funcional, en el cual ciertos parámetros son modificados de acuerdo a la situación en que se encuentren.

En el caso de utilizar partículas, éstas tienen las siguientes propiedades:

- Posición inicial.
- Velocidad inicial.
- Tamaño inicial.
- Calor inicial.
- Transparencia inicial.
- Forma.
- Tiempo de vida.

Por ejemplo, en un determinado momento t un grupo de partículas N puede estar regido por:

$$N_t = M_t + Random(V)\, V_t$$

M_t= Población inicial.
N_t= Variación de la población.

Ejemplo Utilizando OpenGl

Opengl soporta texturas en una y dos dimensiones. Una textura en una dimensión es una imagen con anchura pero sin altura, o viceversa; tienen un único píxel de ancho o de alto. La textura de dos dimensiones es una imagen con más de un píxel de alto o ancho y se abre generalmente con un fichero .BMP.

Todas estas texturas se componen de valores de color RGB y pueden incluir valores alfa (de transparencia). Las texturas siguen las mismas reglas de almacenamiento que los mapas de bits.

Para definir una textura lo hacemos con las siguientes instrucciones:

*void glTextImage1D(GLenum target, GLint level, GLint components, GLsizei width, GLint border, GLenum format, GLenum tipo, const GLvoid *pixels);*

void glTexImage2D(GLenum *target*, GLint *level*, GLint *components*, GLsizei *width*, GLsizei *height*, GLint *border*, GLenum *format*, GLenum *type*, const GLvoid **pixels*);

Donde :

- *target* debe valer GL_TEXTURE_2D
- *level* indica el nivel de detalle de la textura (normalmente 0.)
- *components* indica el nº de componentes del color. Usualmente se usan componentes RGB, y especificaremos 3. Pero también se pueden hacer texturas semitransparentes, con lo que se utiliza un formato RGBA (4 componentes). En ese caso indicaríamos un valor de 4.
- *width* indica el ancho de la imagen de la textura. Debe ser una potencia de 2.
- *height* indica el alto de la imagen de la textura. Debe ser una potencia de 2.
- *border* indica si se utiliza un borde en la textura (1) o no (0). Usualmente es 0.
- *format* indica el formato del valor de cada píxel. Normalmente se utiliza GL_RGB.
- *type* indica el tipo de datos usado para cada componente del valor de un píxel. Puede ser uno de los siguientes valores: GL_UNSIGNED_BYTE, GL_BYTE, GL_BITMAP, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT o GL_FLOAT.

- *pixels* es un puntero al mapa de valores de los pixels. Es la imagen en si.

Los argumentos ancho y alto deben ser potencia de 2. La textura la podemos definir mediante la lectura de un mapa de bits o podemos definir generar una textura en función de generar una matriz conteniendo los tres planos RGB de ella, como por ejemplo:

```
void textura(void)
{
int i, j, c;
      for (i = 0; i < dataHeight; i++) {
      for (j = 0; j < dataWidth; j++) {
      c = ((((i&0x8)==0)^((j&0x8))==0))*255;
            data[i][j][0] = (GLubyte) c;
            data[i][j][1] = (GLubyte) c;
            data[i][j][2] = (GLubyte) c;
      }
      }
}
```

Una vez definida y cargada la textura debemos:

- Habilitar el mapeado de texturas

  ```
  glEnable(GL_TEXTURE_2D);
  ```

- Especificar que imagen va a ser utilizada como textura

  void glTexImage2D(GLenum *target*, GLint *level*, GLint *components*, GLsizei *width*, GLsizei *height*, GLint *border*, GLenum *format*, GLenum *type*, const GLvoid **pixels*);

- Mapear la textura, cuando se esta dibujando el objeto, hay que indicar, para cada vértice de este, que posición de la textura le corresponde. Esto se hace mediante la siguiente función :

  ```
  void glTexCoord2f( GLfloat s, GLfloat t);
  ```

  Donde (s,t) indica una posición sobre el mapa de la imagen.

  Lo que se hace es indicar la coordenada de la textura antes de indicar el vértice del polígono, Por ejemplo para un dibujo de un cuadrado

  ```
  void Cuadrado(void)
  {
        glBegin(GL_QUADS);
        glTexCoord2f(0.0,1.0);glVertex3f(-1.0,1.0,0.0);
        glTexCoord2f(1.0,1.0);glVertex3f(1.0,1.0,0.0);
        glTexCoord2f(1.0,0.0);glVertex3f(1.0,-1.0,0.0);
  ```

```
            glTexCoord2f(0.0,0.0);glVertex3f(-1.0,-1.0,0.0);
        glEnd();
        }
```

- Indicar como la textura va a ser aplicada a cada píxel, aquí hay varios puntos que indicar. El primero de ellos es indicar que ocurre con el tamaño de las texturas. Cuando uno referencia las coordenadas de las texturas, se indican valores entre 0 y 1, que dan los limites de las texturas. Cuando uno referencia un valor mayor que 1 o menor que 0, se esta fuera del mapa de la imagen. ¿Que hacer en estos casos?. Hay dos posibilidades. La primera es repetir los pixels de los bordes de la textura cuando se referencie fuera de ella, lo cual no parece que tenga mucha utilidad. La otra posibilidad es la de repetir la textura. Esto es, en lugar de tener un mapa con solo una imagen, se tiene un mapa donde la imagen de la textura esta repetida infinitas veces, unas contiguas a las otras.

Para indicar si se quiere repetir el borde de la textura, o se quiere repetir la textura completa se utiliza la siguiente función:

```
void glTexParameterf( GLenum target, GLenum pname, GLfloat param );
```

Donde :

- target debe ser GL_TEXTURE_2D.
- pname puede ser GL_TEXTURE_WRAP_S o GL_TEXTURE_WRAP_T, donde el primero indica las coordenadas X de la textura, y el segundo las coordenadas Y.
- param indica si queremos que se repita el borde de la textura (GL_CLAMP) o si queremos que se repita la textura completa (GL_REPEAT).

El siguiente ejemplo muestra una escena de los ejemplos anteriores donde aplicamos la textura generada por programa (tablero de ajedrez) y esta es mapeada a una esfera

El listado de las funciones más relevantes son:

```
void __fastcall TForm1::OglWindowOglResize(TObject *Sender)
{
      Camera.OnSetViewport((GLshort)OglWindow->Width,
(GLshort)OglWindow->Height);
      glViewport(0,0,OglWindow->Width,OglWindow->Height);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      Camera.OglPerspectiveTransformation();
      glMatrixMode(GL_MODELVIEW);
     glLoadIdentity();
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::OglWindowOglCreate(TObject *Sender)
{
     glShadeModel(GL_SMOOTH);
     glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
     glClearDepth(1.0f);
     glEnable(GL_DEPTH_TEST);
     glDepthFunc(GL_LEQUAL);
     glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
     Sphere = gluNewQuadric();
     spot = gluNewQuadric();
     x->Position=lightPos[0];
     y->Position=lightPos[1];
     z->Position=lightPos[2];
     glClearColor (0.0, 0.0, 0.0, 0.0);
     glShadeModel(GL_FLAT);
     glEnable(GL_DEPTH_TEST);
     makeCheckImage();
     glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
     glGenTextures(1, &texName);
     glBindTexture(GL_TEXTURE_2D, texName);
     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
     glTexImage2D(GL_TEXTURE_2D,                    0,                    GL_RGB,
     checkImageWidth,checkImageHeight, 0, GL_RGB, GL_UNSIGNED_BYTE,
     checkImage);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::OglWindowOglPaint(TObject *Sender)
{
glPushMatrix();
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
       Camera.OglViewTransformation();
       iluminacion();
       if(t==1) glEnable(GL_TEXTURE_2D);
       glBindTexture(GL_TEXTURE_2D, texName);
       glRotatef(angulo,1,1,0);
       glPushMatrix();
            if(t==1) glEnable(GL_LIGHTING);
            glColor3f(1.0f, 0.0f, 0.0f);
            gluQuadricDrawStyle(Sphere, GLU_FILL);
            gluQuadricNormals(Sphere, GLU_SMOOTH);
            gluQuadricTexture(Sphere,GL_TRUE);
            gluSphere(Sphere, 1, 50, 50);
            glColor3f(0.0f, 0.0f, 1.0f);
            glEnable(GL_LIGHTING);
            glBegin(GL_QUADS);
              glNormal3f(0,1,0);
               glVertex3f(-lado,pos,-lado);
               glVertex3f(-lado,pos,lado);
               glVertex3f(lado,pos,lado);
               glVertex3f(lado,pos,-lado);
             glEnd();
            if(l==1){
```

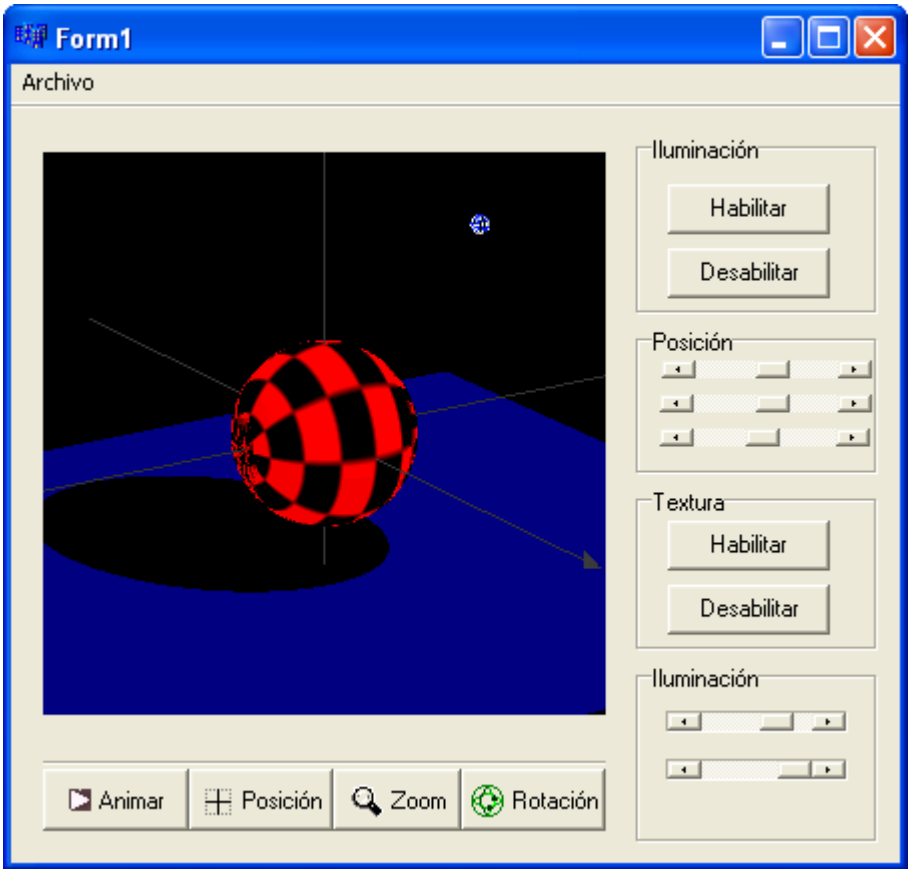
```
                glDisable(GL_LIGHTING);
                 if(t==1)     glDisable(GL_TEXTURE_2D);
                glColor4f(0.0,0.0,0.0,0.9);
                myShadowMatrix(floor_equation,lightPos);
                gluSphere(Sphere, 1, 50, 50);
                glEnable(GL_LIGHTING);
                glEnable(GL_TEXTURE_2D);
              }

        glPopMatrix();
        ejes();
        if(l==1) puntos();
        glDisable(GL_TEXTURE_2D);

glPopMatrix();
}
//---------------------------------------------------------------------------


void __fastcall TForm1::FormCreate(TObject *Sender)
{
  OglCamera::LoadCursors();
  Camera.OglWindow = OglWindow;
  Camera.SetTarget( 0.0, 0.0, 0.0);
  Camera.SceneRadius = 1.5;
  Camera.Twist = 0.0;
  Camera.ZoomMin = 0.1; Camera.ZoomMax = 5;
  Camera.Zoom  = 0.5;
}
void makeCheckImage(void)
{
int i, j, c;
        for (i = 0; i < checkImageHeight; i++) {
        for (j = 0; j < checkImageWidth; j++) {
        c = ((((i&0x8)==0)^((j&0x8))==0))*255;
                checkImage[i][j][0] = (GLubyte) c;
                checkImage[i][j][1] = (GLubyte) c;
                checkImage[i][j][2] = (GLubyte) c;
        }
        }
}
```

El resultado visual se muestra en la siguiente figura:



El listado completo de las funciones utilizadas es el siguiente:

```
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::OglWindowOglResize(TObject *Sender)
{
        Camera.OnSetViewport((GLshort)OglWindow->Width,
(GLshort)OglWindow->Height);
        glViewport(0,0,OglWindow->Width,OglWindow->Height);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        Camera.OglPerspectiveTransformation();
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::OglWindowOglCreate(TObject *Sender)
{
```

```
      glShadeModel(GL_SMOOTH);
      glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
      glClearDepth(1.0f);
      glEnable(GL_DEPTH_TEST);
      glDepthFunc(GL_LEQUAL);
      glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
      Sphere = gluNewQuadric();
      spot = gluNewQuadric();
      x->Position=lightPos[0];
      y->Position=lightPos[1];
      z->Position=lightPos[2];
      glClearColor (0.0, 0.0, 0.0, 0.0);
      glShadeModel(GL_FLAT);
      glEnable(GL_DEPTH_TEST);
      makeCheckImage();
      glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
      glGenTextures(1, &texName);
      glBindTexture(GL_TEXTURE_2D, texName);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,   GL_REPEAT);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
      glTexImage2D(GL_TEXTURE_2D,                    0,                    GL_RGB,
      checkImageWidth,checkImageHeight, 0, GL_RGB, GL_UNSIGNED_BYTE,
       checkImage);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::OglWindowOglPaint(TObject *Sender)
{
glPushMatrix();
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
       Camera.OglViewTransformation();
       iluminacion();
       if(t==1) glEnable(GL_TEXTURE_2D);
       glBindTexture(GL_TEXTURE_2D, texName);
       glRotatef(angulo,1,1,0);
       glPushMatrix();
              if(t==1) glEnable(GL_LIGHTING);
            glColor3f(1.0f, 0.0f, 0.0f);
            gluQuadricDrawStyle(Sphere, GLU_FILL);
            gluQuadricNormals(Sphere, GLU_SMOOTH);
            gluQuadricTexture(Sphere,GL_TRUE);
            gluSphere(Sphere, 1, 50, 50);
            glColor3f(0.0f, 0.0f, 1.0f);
            glEnable(GL_LIGHTING);
            glBegin(GL_QUADS);
              glNormal3f(0,1,0);
                        glVertex3f(-lado,pos,-lado);
                        glVertex3f(-lado,pos,lado);
                        glVertex3f(lado,pos,lado);
                        glVertex3f(lado,pos,-lado);
              glEnd();
            if(l==1){
                  glDisable(GL_LIGHTING);
                  if(t==1)     glDisable(GL_TEXTURE_2D);
```

```
                    glColor4f(0.0,0.0,0.0,0.9);
                    myShadowMatrix(floor_equation,lightPos);
                    gluSphere(Sphere, 1, 50, 50);
                    glEnable(GL_LIGHTING);
                   glEnable(GL_TEXTURE_2D);
              }

       glPopMatrix();
       ejes();
       if(l==1) puntos();
       glDisable(GL_TEXTURE_2D);

glPopMatrix();
}
//---------------------------------------------------------------------------


void __fastcall TForm1::FormCreate(TObject *Sender)
{
  OglCamera::LoadCursors();
  Camera.OglWindow = OglWindow;
  Camera.SetTarget( 0.0, 0.0, 0.0);
  Camera.SceneRadius = 1.5;
  Camera.Twist = 0.0;
  Camera.ZoomMin = 0.1; Camera.ZoomMax = 5;
  Camera.Zoom  = 0.5;
}
//---------------------------------------------------------------------------

void ejes(void)
{
       glColor3f(0.4f, 0.4f, 0.4f);
     glBegin(GL_LINES);
        glVertex3f(5.0f,0.0f, 0.0f);
        glVertex3f(-5.0f,0.0f, 0.0f);
     glEnd();

              glBegin(GL_LINES);
        glVertex3f(0.0f,5.0f, 0.0f);
        glVertex3f(0.0f,-5.0f, 0.0f);
     glEnd();

              glBegin(GL_LINES);
        glVertex3f(0.0f,0.0f, 5.0f);
        glVertex3f(0.0f,0.0f, -5.0f);
     glEnd();

       glBegin(GL_TRIANGLES);
             glVertex3f(5.0f,0.0f,0.0f);
             glVertex3f(4.7f,0.1f,0.0f);
             glVertex3f(4.7f,-0.1,0.0);
       glEnd();
      glBegin(GL_TRIANGLES);
```

```
            glVertex3f(0.0f,5.0f,0.0f);
            glVertex3f(-0.1f,4.7f,0.0f);
            glVertex3f(0.1f,4.7f,0.0f);
        glEnd();
        glBegin(GL_TRIANGLES);
            glVertex3f(0.0f,0.0f,-5.0f);
            glVertex3f(0.0f,0.1f,-4.7f);
            glVertex3f(0.0f,-0.1f,-4.7f);
        glEnd();
}

   void iluminacion(void)
{
      glEnable(GL_DEPTH_TEST);
      glFrontFace(GL_CCW);
      glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
      glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
      glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
      glEnable(GL_LIGHT0);
      glEnable(GL_COLOR_MATERIAL);
     glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f );
}

void puntos(void)
{
   glPushMatrix();
      glColor3f(1.0f, 1.0f, 1.0f);
      glTranslatef( lightPos[0],lightPos[1], lightPos[2]);
            gluQuadricDrawStyle(Sphere, GLU_FILL);
            gluQuadricNormals(Sphere, GLU_SMOOTH);
            gluSphere(Sphere, 0.1, 50, 50);
   glPopMatrix();
}
void myShadowMatrix(float ground[4], float light[4])
{
  float dot;
  float shadowMat[4][4];

  dot = ground[0] * light[0] +
    ground[1] * light[1] +
    ground[2] * light[2] +
    ground[3] * light[3];

  shadowMat[0][0] = dot - light[0] * ground[0];
  shadowMat[1][0] = 0.0 - light[0] * ground[1];
  shadowMat[2][0] = 0.0 - light[0] * ground[2];
  shadowMat[3][0] = 0.0 - light[0] * ground[3];

  shadowMat[0][1] = 0.0 - light[1] * ground[0];
  shadowMat[1][1] = dot - light[1] * ground[1];
  shadowMat[2][1] = 0.0 - light[1] * ground[2];
  shadowMat[3][1] = 0.0 - light[1] * ground[3];
```

```
  shadowMat[0][2] = 0.0 - light[2] * ground[0];
  shadowMat[1][2] = 0.0 - light[2] * ground[1];
  shadowMat[2][2] = dot - light[2] * ground[2];
  shadowMat[3][2] = 0.0 - light[2] * ground[3];

  shadowMat[0][3] = 0.0 - light[3] * ground[0];
  shadowMat[1][3] = 0.0 - light[3] * ground[1];
  shadowMat[2][3] = 0.0 - light[3] * ground[2];
  shadowMat[3][3] = dot - light[3] * ground[3];

  glMultMatrixf((const GLfloat *) shadowMat);
}


void makeCheckImage(void)
{
int i, j, c;
        for (i = 0; i < checkImageHeight; i++) {
        for (j = 0; j < checkImageWidth; j++) {
        c = ((((i&0x8)==0)^((j&0x8))==0))*255;
                checkImage[i][j][0] = (GLubyte) c;
                checkImage[i][j][1] = (GLubyte) c;
                checkImage[i][j][2] = (GLubyte) c;
        }
        }
}
```