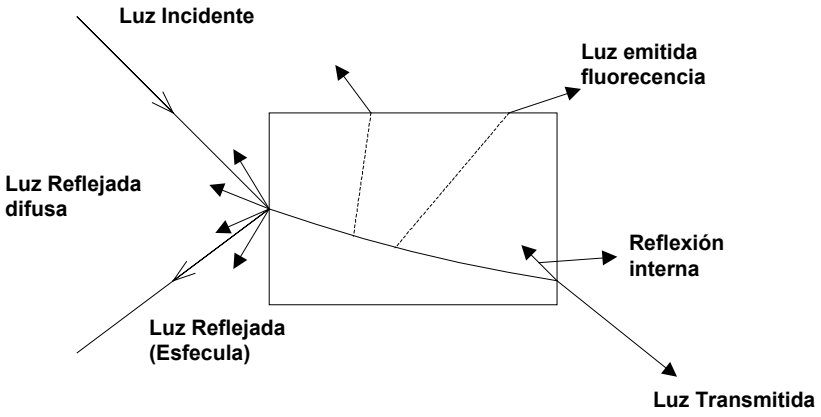
CAPITULO 5

Modelo Básico de Iluminación

Se describe un modelo básico de iluminación, en el cual se hacen suposiciones que permiten simplificar el problema, y aún así tener un efecto visual aceptable de la escena.

5.1 **Modelo Básico**

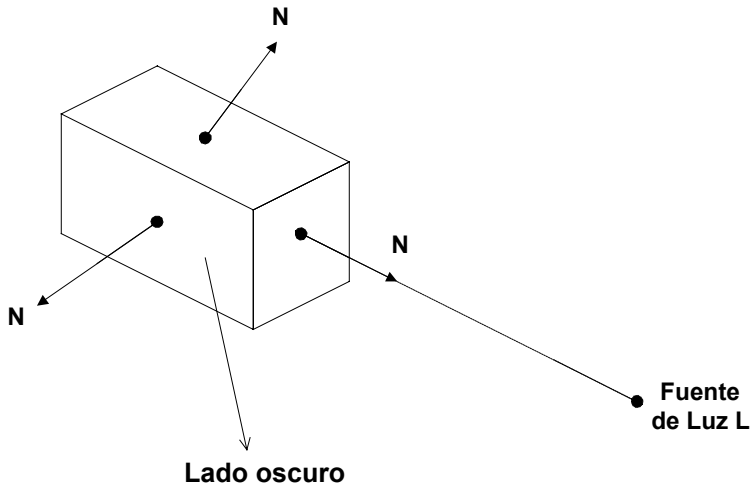
Este modelo se establece como sigue:



5.1.1 Reflexión Difusa

La reflexión difusa está regida por la ley de Lambert. La luz difusa reflejada depende del ángulo formado por la luz incidente y la normal a la cara iluminada.

Ej:



$$Id = Ii * Kd(L * N)$$

L y *N* tienen módulo 1, en el caso que hay varias fuentes de iluminación.

$$Id = Kd\sum_{n} Ii_{1n}(L_n * N)$$ Para una cara cualquiera.

5.1.2 Luz Ambiente

Esta luz, producto de múltiples reflexiones en una escena permite que los objetos no tengan caras totalmente oscuras.

Así la luz ambiente se interpreta sólo por una variable constante en la ecuación que determinamos, esto es:

$$I = Ia * Ka + Ii * Kd(L * N)$$

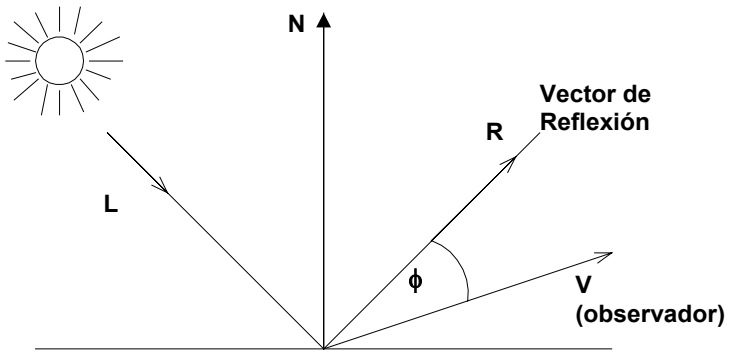
5.1.3 Distancia

La distancia de la fuente de luz, afecta a la intensidad reflejada, pero en general esto se introduce en la ecuación sólo por una constante de división. Esto es:

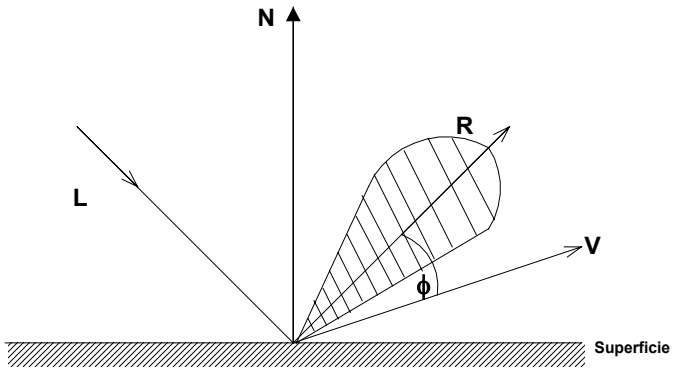
$$I = Ia * Ka + Id * Kd(L * N)/d \qquad d = factor\ dis\tan cia$$

5.1.4 Reflexión Especular

La reflexión especular permite observar brillos en los objetos en alguna parte específica de éstas, dependiendo de la rugosidad de la superficie. En un espejo perfecto, esta reflexión es total. Esta reflexión se puede representar de la siguiente forma:



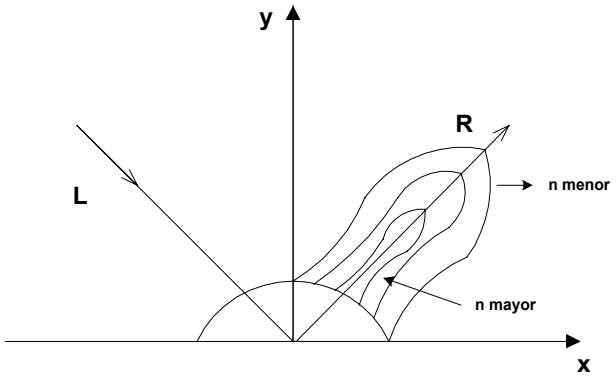
Si la reflexión es perfecta y el ángulo ϕ es $\neq 0$, entonces el observador no ve luz. Sin embargo, la reflexión nunca es perfecta y en realidad tiene la siguiente forma (aproximada).

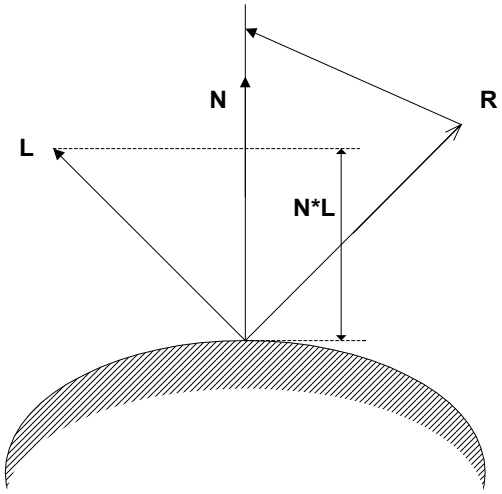


5.2 Intensidad y Reflexión

Bui-tuong Phong representó esta función por la fórmula $\cos^n \phi$. De esta manera ahora podemos representar la intensidad *I* al incluir la reflexión especular como:

$$I = Ra * Ia + \left[kd * Id(L * N)\right] + IiKs * \cos^n \phi \quad / d$$
$$I = Ra * Ia + \left[kd * Id(L * N)\right] + IiKs * (R * V)^n \quad / d$$





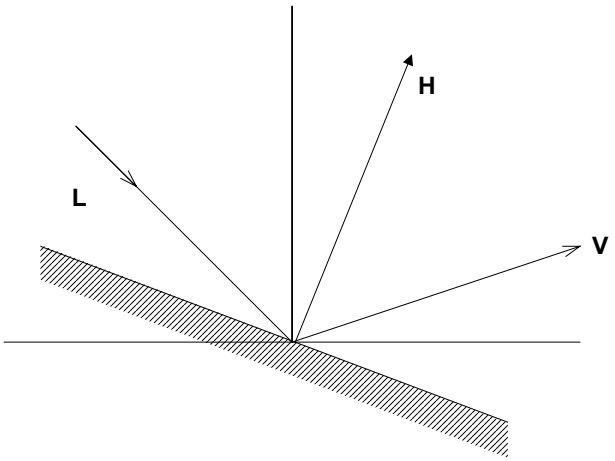
$$R + L = (2N * L)N$$

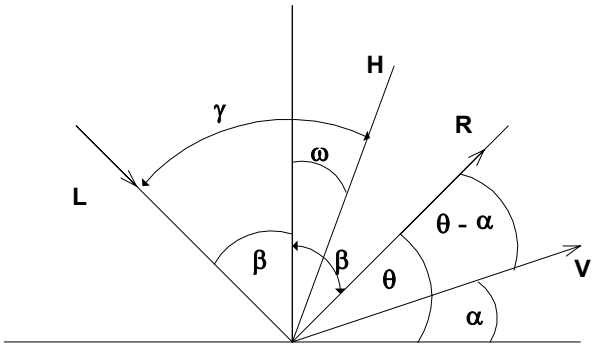
Luego

$$R = (2N * L)N - L$$

Aunque el valor de *N* se puede calcular, esto es computacionalmente caro. Para disminuir este gasto computacional se introduce otro vector *H* de la siguiente manera.

Se define $H = \frac{L+V}{|L+V|}$ como la normal a una superficie cuya reflexión máxima es a través de *V*.





$$\gamma = \frac{2\beta + \theta - \alpha}{2} \qquad \angle HR = 90^\circ - \omega - \theta$$

$$\angle LH = \angle HV \qquad \beta = 90^\circ - \theta$$

$$\gamma = \angle HR + \theta - \alpha$$

$$\gamma = 90^\circ - \omega - \theta + \theta - \alpha = 90^\circ - \omega - \alpha$$

Igualando:

$$2\beta + \theta - \alpha = 180^\circ - 2\omega - 2\alpha$$
$$2(90^\circ - \theta) + \theta - \alpha = 180^\circ - 2\omega - 2\alpha$$
$$180^\circ - 2\theta + \theta - \alpha = 180^\circ - 2\omega - 2\alpha$$
$$-\theta - \alpha = -2\omega - 2\alpha$$
$$-\theta + \alpha = -2\omega$$
$$\omega = \frac{\theta - \alpha}{2}$$

Es decir:

$$N * H = \frac{R * V}{2}$$

Este efecto es compensado al ajustar *N*.

$$I = Ia * Ka + \left[Ii * Kd(L * N) + Ks(N * H)^n \right] / d$$

La fórmula anterior debe establecerse para cada color *R,G,B* de una pantalla de este tipo. Es decir deben encontrarse I_r, I_g, I_b.

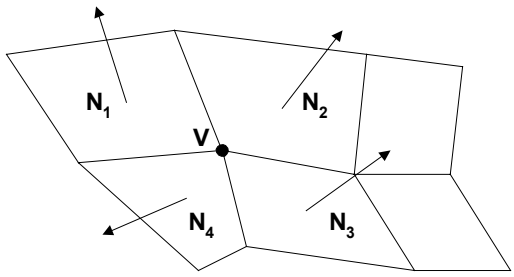
5.3 Pintado de Gouraud

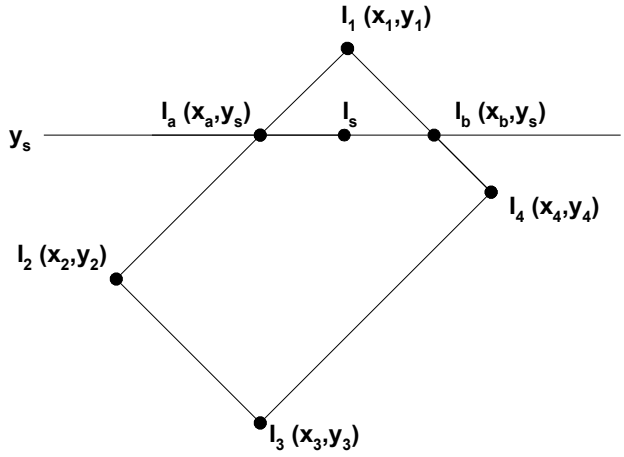
Este es un esquema en el que se interpolan las intensidades a través de la superficie de un polígono.

Las fases a realizar son:

- Determine el vector normal unitario promedio en cada vértice del polígono.
- Aplique un modelo de iluminación a cada vértice para calcular su intensidad.
- Interpole linealmente las intensidades de los vértices sobre la superficie del polígono.

$$N_v = \frac{\sum_{k=1}^{n} N_k}{\left| \sum_{k=1}^{n} N_k \right|}$$





$$Ia = t * I_1 + (1-t) * I_2 \qquad t \in [0,1]$$
$$ys = t * y_1 + (1-t) * y_2$$
$$t = \frac{ys - y_2}{y_1 - y_2}$$

Reemplazando en I_a:

$$Ia = \frac{ys - y_2}{y_1 - y_2} * I_1 + \left(1 - \frac{ys - y_2}{y_1 - y_2}\right) * I_2$$
$$Ia = \frac{ys - y_2}{y_1 - y_2} * I_1 + \left(\frac{y_1 - y_2 - ys + y_2}{y_1 - y_2}\right) * I_2$$
$$Ia = \frac{1}{y_1 - y_2} * \left[I_1(ys - y_2) + I_2(y_1 - ys)\right]$$

De igual manera:

$$Ib = \frac{1}{y_1 - y_4} * \left[I_1(ys - y_4) + I_4(y_1 - ys)\right]$$
$$Is = \frac{1}{x_b - x_a} * \left[Ia(x_b - x_s) + Ib(x_s - x_a)\right]$$

Sea $x_s = \Delta x + x_{s-1}$

$$Is = \frac{1}{x_b - x_a} * \left[Ia\left(x_b - \Delta x - x_{s-1}\right) + Ib\left(\Delta x - x_{s-1} - x_a\right)\right]$$

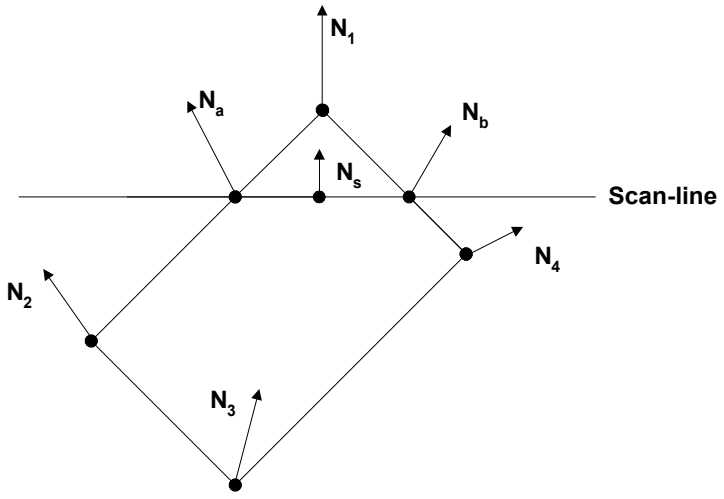
$$Is = \frac{1}{x_b - x_a} * \left[Iax_b - Ia\Delta x - Iax_{s-1} + Ib\Delta x - Ibx_{s-1} - Ibx_a \right]$$

$$Is = \frac{1}{x_b - x_a} * \left[\Delta x(Ib - Ia) + Ia(x_b - x_{s-1}) + Ib(x_{s-1} - x_a)\right]$$

$$Is = \underbrace{\frac{\Delta x}{x_b - x_a}(Ib - Ia)}_{\Delta Is} + \underbrace{\frac{1}{x_b - x_a}\left[Ia(x_b - x_{s-1}) + Ib(x_{s-1} - x_a)\right]}_{I_{s-1}}$$

5.4 Pintado de Pong

En este esquema se utilizan las normales para cada punto donde se calcula la intensidad. Luego a partir de las normales promedio de los vértices, se interpolan las normales para obtener su valor en cualquier punto de la orilla o línea de scan-line.



$$Na = \frac{1}{y_1 - y_2} * \left[N_1 (ys - y_2) + N_2 (y_1 - ys) \right]$$

$$Nb = \frac{1}{y_1 - y_4} * \left[N_1 (ys - y_4) + N_4 (y_1 - ys) \right]$$

$$Ns = \frac{1}{x_b - x_a} * \left[Ns(x_b - x_s) + N_b (x_s - x_a) \right]$$

Ejemplo Utilizando OpenGl

OpenGl soporta varios tipos de iluminación que permiten dar a una escena un importante realismo. OpenGL permite crear y parametrizar distintas fuentes de luces y activarlas o desactivarlas cuando el usuario lo requiera. Para definir una fuente de luz definimos las siguientes instrucciones:

```
glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
```

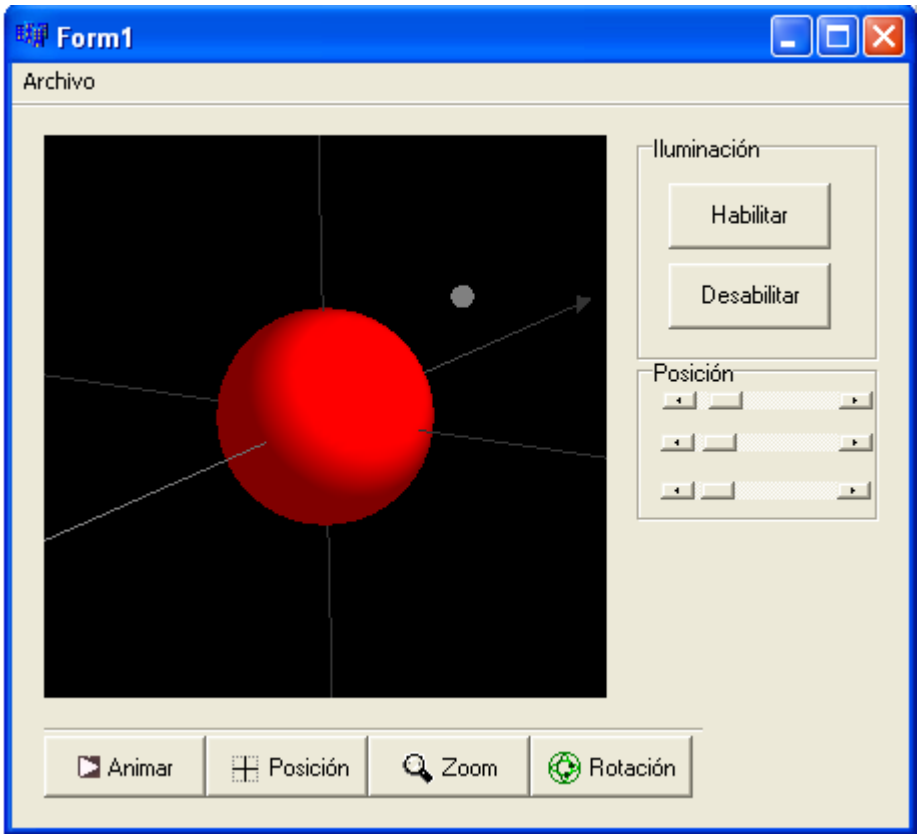
Estas 3 instrucciones nos permiten definir la luz 0 con componentes de luz ambiente, difusa y posición de la luz incidente sobre nuestros objetos. También podemos agregar cualidades como:

```
glEnable(GL_DEPTH_TEST);     // Eliminación de caras ocultas
glFrontFace(GL_CCW);         // Polígonos de sentido positivo    hacia afuera
glEnable(GL_COLOR_MATERIAL);  // Activa el seguimiento de color
// Selecciona las propiedades de material para el seguimiento
//  de los   valores de glColor
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
// Fondo iluminado de negro
glClearColor(0.0f, 0.0f, 0.0f, 1.0f );
```

y finalmente activamos o desactivamos la fuente de luz mediante

```
glEnable(GL_LIGHT0);
glDisable(GL_LIGHT0);
```

La siguiente figura muestra una esfera iluminada con una fuente de luz con componentes de luz difusa y ambiental



El código principal de nuestro programa es el siguiente:

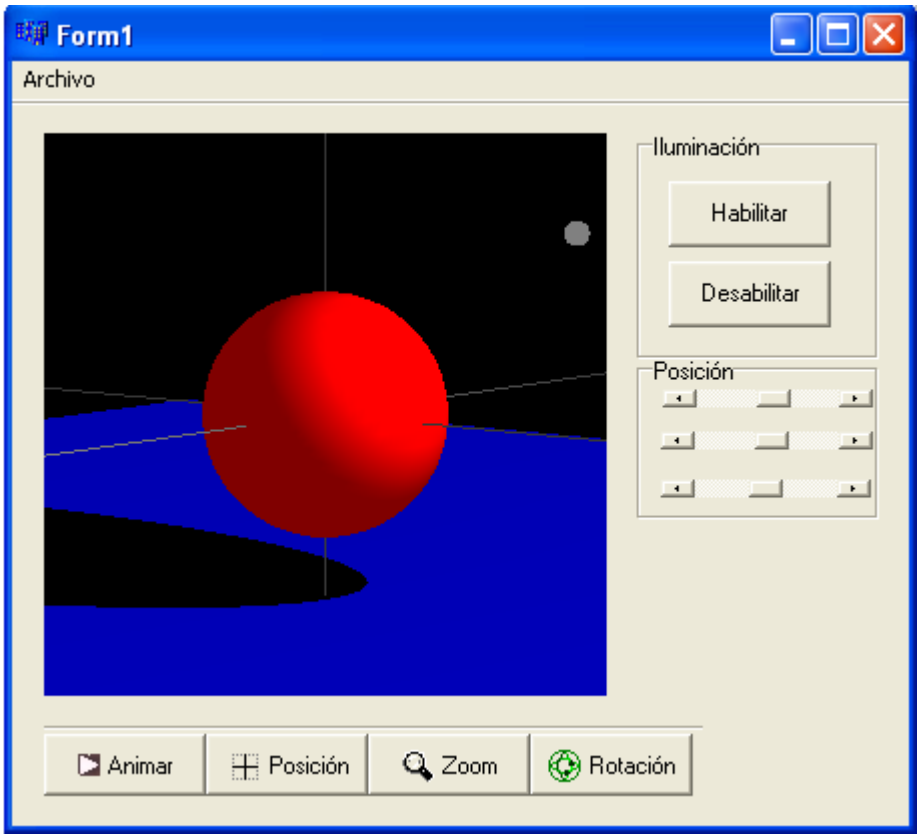
```
void __fastcall TForm1::OglWindowOglResize(TObject *Sender)
{
      Camera.OnSetViewport((GLshort)OglWindow->Width,
(GLshort)OglWindow->Height);
      glViewport(0,0,OglWindow->Width,OglWindow->Height);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      Camera.OglPerspectiveTransformation();
      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=5;
    y->Position=5;
    z->Position=5;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::OglWindowOglPaint(TObject *Sender)
{
    glPushMatrix();
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    Camera.OglViewTransformation();
    iluminacion();
    glRotatef(angulo,1,1,0);
   //   ESFERA

     glPushMatrix();
           glColor3f(1.0f, 0.0f, 0.0f);
           gluQuadricDrawStyle(Sphere, GLU_FILL);
           gluQuadricNormals(Sphere, GLU_SMOOTH);
           gluSphere(Sphere, 1, 50, 50);
      glPopMatrix();
      ejes();
      if(l==1) puntos();
    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);    // Eliminación de caras ocultas
     glFrontFace(GL_CCW);         // Polígonos de sentido positivo hacia afuera

     // Define e inicia la luz 0
     glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
     glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
     glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
     glEnable(GL_LIGHT0);

     // Activa el seguimiento de color
     glEnable(GL_COLOR_MATERIAL);

     // Selecciona las propiedades de material para el seguimiento
    // de los valores de glColor
     glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
```

```
    // Fondo iluminado de azul
    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();
}
```

Un efecto importante en toda escena es el manejo de sombras. Opengl no maneja directamente la creación de sombras, sin embargo, podemos proyectar los objetos de una escena en algún plano en la dirección de la fuente de luz para generar las sombras que proyectarían nuestros objetos. La figura siguiente muestra la misma esfera anterior donde hemos utilizado un plano de proyección para calcular la sombra que proyectaría nuestra esfera en la dirección de la fuente de luz.



El listado de las funciones más importantes y el cálculo de la proyección se describen a continuación.

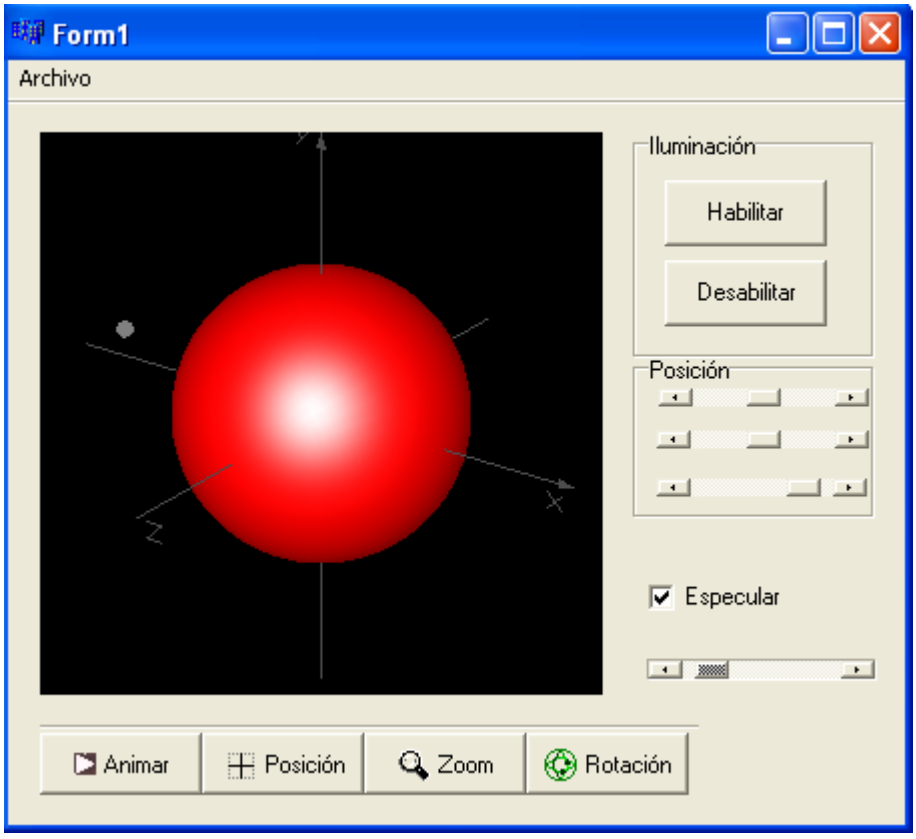
```
void __fastcall TForm1::OglWindowOglResize(TObject *Sender)
{
      Camera.OnSetViewport((GLshort)OglWindow->Width,
(GLshort)OglWindow->Height);
      glViewport(0,0,OglWindow->Width,OglWindow->Height);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      Camera.OglPerspectiveTransformation();
      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];
}
//---------------------------------------------------------------------------
void __fastcall TForm1::OglWindowOglPaint(TObject *Sender)
{
    glPushMatrix();
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    Camera.OglViewTransformation();
    iluminacion();
    glRotatef(angulo,1,1,0);
    glPushMatrix();
          glColor3f(1.0f, 0.0f, 0.0f);
          gluQuadricDrawStyle(Sphere, GLU_FILL);
          gluQuadricNormals(Sphere, GLU_SMOOTH);
          gluSphere(Sphere, 1, 50, 50);
          glBegin(GL_QUADS);
             glColor3f(0.0f, 0.0f, 1.0f);
            // Indicamos la normal de suelo para la iluminación
            glNormal3fv(floor_normal);
            glVertex3f(-SIZEFLOOR,HEIGHTFLOOR,-SIZEFLOOR);
                  glVertex3f(SIZEFLOOR,HEIGHTFLOOR,-SIZEFLOOR);
                  glVertex3f(SIZEFLOOR,HEIGHTFLOOR,SIZEFLOOR);
                  glVertex3f(-SIZEFLOOR,HEIGHTFLOOR,SIZEFLOOR);
            glEnd();
```

```
            //  aquí proyección de sombra
            if(l==1){
              glDisable(GL_LIGHTING);
              glColor4f(0.0,0.0,0.0,0.9);
              myShadowMatrix(floor_equation,lightPos);
              gluSphere(Sphere, 1, 50, 50);
              glEnable(GL_LIGHTING);
            }

      glPopMatrix();
      ejes();
      if(l==1) puntos();
     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);
}
```

En el ejemplo anterior no se ha considera reflexión especular ni las propiedades del material que está iluminado. Opengl permite definir estos aspectos mediante propiedades específicas de reflexión y de material. Al ejemplo anterior de la esfera le hemos agregado ahora componentes de reflexión especular y propiedades del material, el resultado puede verse en la siguiente figura.



El listado de las funciones más importantes es el siguiente:

```
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=5;
     y->Position=5;
     z->Position=5;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::OglWindowOglPaint(TObject *Sender)
{
     glPushMatrix();
     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     Camera.OglViewTransformation();
     if( l==1    )      iluminacion();

//       Todos los materiales siguientes tienen reflexión especular completa
//      con alto brillo
     glMaterialfv(GL_FRONT, GL_SPECULAR, specref2);
     glMateriali(GL_FRONT, GL_SHININESS,0);
     if(Especular->Checked){
       glMaterialfv(GL_FRONT, GL_SPECULAR, specref1);
       glMateriali(GL_FRONT, GL_SHININESS,Brillo->Position);
     }
   glRotatef(angulo,1,1,0);
   //    ESFERA

       glPushMatrix();
             glColor3f(1.0f, 0.0f, 0.0f);
             gluQuadricDrawStyle(Sphere, GLU_FILL);
             gluQuadricNormals(Sphere, GLU_SMOOTH);
             gluSphere(Sphere, 2.5, 50, 50);
       glPopMatrix();
```

```
        ejes();
        if(l==1) puntos();
      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;
}
//---------------------------------------------------------------------------
```