

5. ILUMINANDO LA ESCENA

En este capítulo se explican las funciones que OpenGL pone a disposición para manejar la iluminación de la escena y a crear, posicionar y dar propiedades a diferentes tipos de luces. La posición de las luces se realiza mediante el ratón con funciones similares a las utilizadas para posicionar la cámara en el modo examinar.

5.1. AÑADIENDO ILUMINACIÓN A LA ESCENA

Los pasos requeridos para añadir iluminación a la escena son los siguientes:

1. **Definir los vectores normales para cada vértice de cada objeto:** estas normales determinan la orientación de la superficie del objeto relativa a las fuentes de luz.
2. **Crear, posicionar y activar una o más fuentes de luz**
3. **Definir las propiedades del material para los objetos en la escena.**

La presente práctica se va a centrar exclusivamente en el segundo punto. Se utilizará como objeto una tortuga en 3D realizada con esferas, de forma que las normales para dichas esferas viene definidas como parte de la rutina **glutSolidSphere()**. Las propiedades del material de los objetos se realizará asignando directamente un color en lugar de definir materiales y posteriormente asignarlos a los objetos.

5.2. CREANDO FUENTES DE LUZ CON OPENGL

5.2.1. Comando **glLight*()**

El comando usado para especificar todas las propiedades luces es **glLight*()**; esta función toma tres argumentos: el primero de ellos identifica la luz para la cual se está especificando una propiedad, la propiedad que se desea especificar y el valor para dicha propiedad. La forma de la función es la siguiente:

```
void glLight{if}(GLenum light, GLenum pname, TYPE param);  
void glLight{if}v(GLenum light, GLenum pname, TYPE *param);
```

| Parameter Name | Default Value | Meaning |
|--------------------------|----------------------|-------------------------------------|
| GL_AMBIENT | (0.0, 0.0, 0.0, 1.0) | Intensidad RGBA de la luz ambiente |
| GL_DIFFUSE | (1.0, 1.0, 1.0, 1.0) | Intensidad RGBA de la luz difusa |
| GL_SPECULAR | (1.0, 1.0, 1.0, 1.0) | Intensidad RGBA de la luz especular |
| GL_POSITION | (0.0, 0.0, 1.0, 0.0) | Posición (x, y, z, w) de la luz |
| GL_SPOT_DIRECTION | (0.0, 0.0, -1.0) | Dirección (x, y, z) del spotlight |
| GL_SPOT_EXPONENT | 0.0 | Spotlight exponent |
| GL_SPOT_CUTOFF | 180.0 | Spotlight cutoff angle |
| GL_CONSTANT_ATTENUATION | 1.0 | Constant attenuation factor |
| GL_LINEAR_ATTENUATION | 0.0 | Linear attenuation factor |
| GL_QUADRATIC_ATTENUATION | 0.0 | Quadratic attenuation factor |

Tabla 5.1. Valores por defecto del Parámetro *pname* de **glLight*()**.

El parámetro *light* que identifica la luz puede tomar los valores `GL_LIGHT0`, `GL_LIGHT1`,..., `GL_LIGHT7`, de forma que se pueden tener hasta 8 luces diferentes simultáneamente en la escena.

El parámetro *pname* controla la propiedad de la luz que se desea especificar. Las diferentes propiedades de la luz que se pueden modificar, su significado, así como los valores por defecto de cada propiedad se muestran en la tabla 5.1.

5.3. INTRODUCIENDO LUCES AL PROGRAMA TECNUNLOGO

5.3.1. Tipos de luces

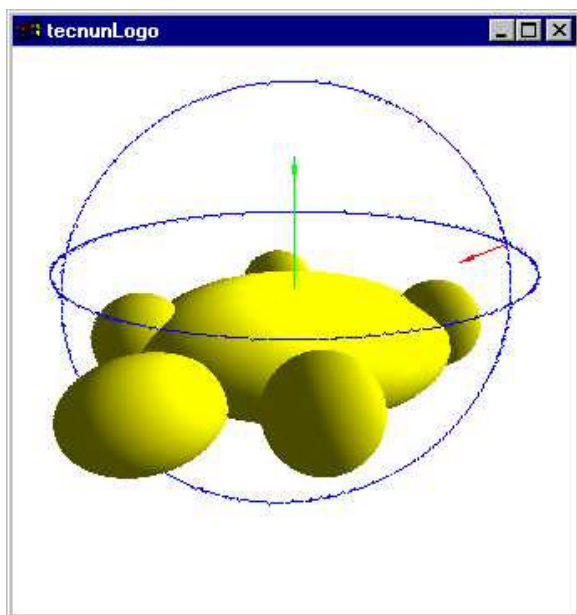


Figura 1. Forma de representar la luz

Se van a introducir tres luces al programa TecnunLogo, todas del mismo tipo. Esto permitirá estudiar cómo varía la iluminación de la escena al cambiar las componentes propias de la luz. Además, permitiendo que puedan ser encendidas o apagadas de forma independiente se pueden estudiar los efectos de combinaciones entre ellas.

En la figura 1 se muestra cómo aparecerán representadas en el programa TecnunLogo las luces. Un paralelo y un meridiano localizan la posición de la luz. En el caso de luz direccional, que es el tipo de luz que se va a introducir, esta viene representada con un vector dirigido siempre hacia el centro de la escena que indica su dirección. En el siguiente capítulo se verán otros tipos de fuentes de luz y sus representaciones.

5.3.2. Interface de usuario

Para que el usuario de la aplicación pueda interactuar con las luces que se van a poner a su disposición, se debe definir un interface de usuario. Este interface constará de una serie de teclas que permitan pasar al Modo Luces y pasar el control de una luz a otra, así como encender y apagar cada una de las luces (Tabla 5.2).

Puesto que a estas alturas el programa ya dispondrá de al menos dos modos de interacción, a saber, modo cámara y modo luces, se va a hacer que para pasar de un modo a otro no se pueda hacer directamente sino que haya que desactivar previamente el modo en curso. Mediante el ratón será posible cambiar la posición de las luces (Tabla 5.3).

| Tecla | Acción |
|-------|--------------------------------------------------------------|
| F1 | Desactivar luces (en general, desactivará cualquier modo) |
| F8 | Modo luces. Cada vez que se pulsa se pasa de una luz a otra. |
| F9 | ON/OFF la luz con la cual se está interactuando. |

Tabla 5.2. Teclas del Modo Luces

| Movimiento del Ratón | Acción |
|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Adelante/Atrás | Movimiento de la luz con la cual se está interactuando a lo largo de un <i>meridiano</i> con centro en el origen de coordenadas de la escena. |
| Izquierda/Derecha | Movimiento de la luz con la cual se está interactuando a lo largo de un <i>paralelo</i> con centro en el eje Y. |
| Adelante/Atrás con botón izquierdo pulsado | Movimiento de la luz a lo largo del vector que une la posición de la luz con el centro de coordenadas de la escena. |

Tabla 5.3. Movimiento de la posición de las luces

5.3.3. Modificaciones en el código

Se va a dotar al programa de tres luces que permitirán en el siguiente capítulo observar las diferencias entre los distintos tipos de luces y su efecto en la escena. Para ello se permitirá interactuar con la posición y dirección de cada una de ellas y mantenerlas encendidas o apagadas independientemente una de otra.

Se necesita definir un interface de luz que guarde los datos de la luz y una serie de funciones auxiliares para el movimiento de las luces. En el proyecto incluiremos los siguientes archivos:

`light.h` contiene las declaraciones del interface de luces y de las funciones de manejo de luces.
`light.c` contiene las definiciones de las funciones para el manejo de las luces.

Además se necesitan una serie de funciones que dibujan primitivas. Entre estas primitivas están una tortuga modelada con esferas, el trazado de meridianos y paralelos en un punto y otras. Se incluyen en el proyecto los siguientes ficheros:

`primitivas.h` contiene las declaraciones de las funciones de dibujo de primitivas.
`primitivas.c` contiene las definiciones de las funciones de dibujo de primitivas.

En la cabecera del fichero `tecnunLogo.c` incluiremos las sentencias:

```
#include "light.h"
#include "primitivas.h"
```

Se debe declarar una variable global que contendrá el interface de cada una de las tres luces, es decir, los datos de cada luz con los cuales alimentaremos a la función que efectivamente realiza los cambios en la luz, `glLight*()`. A su vez se necesita una variable global que indique la luz con la cual se está interactuando en cada momento. Además se definirá una variable global que indique el modo en el que se está trabajando (modo examinar, modo andar, modo luces). Se incluyen las siguientes líneas en el fichero `tecnunLogo.c` inmediatamente después de los ficheros de cabecera:

```
static light **LOCAL_MyLights;
static int current_mode = 0;
static int current_light = -1;
```

Inicialmente se debe crear el interface para las luces y asignárselo a la variable global que se acaba de declarar. Además se deben dar las características a cada una de las luces. De esta forma quedan definidos los interfaces para las tres luces en un array. Esto lo realizamos incluyendo las siguientes sentencias en la función **main()** del programa:

```
int i;
...
//Reservamos memoria para tres interfaces de luces
LOCAL_MyLights = (light **) malloc( 3 * sizeof(light *));
//Creamos las luces y damos a cada una sus características
for(i=0;i<3;i++){
LOCAL_MyLights[i] = CreateDefaultLight();
LOCAL_MyLights[i]->type = AGA_DIRECTIONAL;
LOCAL_MyLights[i]->id = GL_LIGHT0 + i;
LOCAL_MyLights[i]->position[0] = 1.0f;
LOCAL_MyLights[i]->position[1] = 1.0f;
LOCAL_MyLights[i]->position[2] = 1.0f;
LOCAL_MyLights[i]->position[3] = 0.0f;
LOCAL_MyLights[i]->pointAtInfinity[0] = LOCAL_MyLights[0]->position[0];
LOCAL_MyLights[i]->pointAtInfinity[1] = LOCAL_MyLights[0]->position[1];
LOCAL_MyLights[i]->pointAtInfinity[2] = LOCAL_MyLights[0]->position[2];
}
```

OpenGL pone a disposición del programador la posibilidad de asignar colores en lugar de materiales a los objetos que van a ser utilizados en programas que usan iluminación. Esta característica se conoce como **Colour Tracking** y es muy útil pues evita el tener que asignar manualmente las propiedades del material a los objetos cuando la aplicación no requiere tal cosa pues no importan las propiedades del material. Para activarla introducir la siguiente sentencia en la función **main()**:

```
glEnable(GL_COLOR_MATERIAL);
```

La función **SpecialKey()** quedará de la siguiente manera una vez que se incluyen las sentencias necesarias para definir las teclas que permiten pasar al Modo Luces:

```
static void SpecialKey ( int key, int x, int y ){
switch(key) {

case GLUT_KEY_F1:
current_mode = 0;
glutPassiveMotionFunc(MouseMotion);
LOCAL_MyCamera->camMovimiento = CAM_STOP;
current_light = -1;
break;
case GLUT_KEY_F2:
if (current_mode != 0) break;
current_mode = 1;
glutPassiveMotionFunc(Examinar);
LOCAL_MyCamera->camMovimiento = CAM_EXAMINAR;
break;
case GLUT_KEY_F3:
if (current_mode != 0) break;
current_mode = 2;
glutPassiveMotionFunc(MouseMotion);
LOCAL_MyCamera->camMovimiento = CAM_PASEAR;
LOCAL_MyCamera->camAtY = 0;
LOCAL_MyCamera->camViewY = 0;
SetDependentParametersCamera( LOCAL_MyCamera );
break;
case GLUT_KEY_F8:
if (current_mode != 0 && current_mode != 7) break;
current_mode = 7;
if (current_light == -1) glutPassiveMotionFunc(Mouse_Luces);
```

```

        if (current_light != 2) current_light++;
        else current_light = 0;
        printf("Luz actual = %d\n",current_light);
        break;
    case GLUT_KEY_F9:
        if (current_light != -1)
            if ( LOCAL_MyLights[current_light]->switched )
                SwitchLight( LOCAL_MyLights[current_light], FALSE);
            else SwitchLight( LOCAL_MyLights[current_light], TRUE);
        break;
    case GLUT_KEY_HOME: //Reset Camera
        LOCAL_MyCamera->camAtX =0;
        LOCAL_MyCamera->camAtY =0;
        LOCAL_MyCamera->camAtZ =0;
        LOCAL_MyCamera->camViewX = 0;
        LOCAL_MyCamera->camViewY = 1;
        LOCAL_MyCamera->camViewZ = -3;
        SetDependentParametersCamera( LOCAL_MyCamera );
        break;
    default:
        printf("key %d %c X %d Y %d\n", key, key, x, y );
    }
    glutPostRedisplay();
}

```

La función **Mouse_Luces(int x, int y)** moverá la luz con la cual se esté interactuando en ese momento a lo largo de un paralelo o un meridiano según los movimientos del ratón definidos con anterioridad:

```

void Mouse_Luces(int x, int y){
    float rot_x, rot_y;

    rot_y = (float)(old_y - y);
    rot_x = (float)(x - old_x);
    Rotar_Luces_Latitud( LOCAL_MyLights[current_light],rot_y*DEGREE_TO_RAD );
    Rotar_Luces_Longitud( LOCAL_MyLights[current_light], rot_x*DEGREE_TO_RAD );

    LOCAL_MyLights[current_light]->pointAtInfinity[0] =
LOCAL_MyLights[current_light]->position[0];
    LOCAL_MyLights[current_light]->pointAtInfinity[1] =
LOCAL_MyLights[current_light]->position[1];
    LOCAL_MyLights[current_light]->pointAtInfinity[2] =
LOCAL_MyLights[current_light]->position[2];

    old_y = y;
    old_x = x;

    glutPostRedisplay();
}

```

La función **mouse()** quedará de la siguiente forma después de incluir las sentencias necesarias para interactuar con las luces:

```

void mouse(int button, int state, int x, int y){

    old_x = x;
    old_y = y;

    switch(button){
    case GLUT_LEFT_BUTTON:
        if(current_light > 0){
            if (state == GLUT_DOWN)
                glutMotionFunc(Mouse_Luces_Acercar_Alejar);
            if (state == GLUT_UP){
                glutPassiveMotionFunc(Mouse_Luces);
            }
        }
    }
}

```

```

        glutMotionFunc(NULL);
    }
} else {
switch(LOCAL_MyCamera->camMovimiento) {
case CAM_EXAMINAR:
    if (state == GLUT_DOWN) glutMotionFunc(Zoom);
    if (state == GLUT_UP) {
        glutPassiveMotionFunc(Examinar);
        glutMotionFunc(NULL);
    }
    break;
case CAM_PASEAR:
    if (state == GLUT_DOWN) glutMotionFunc(Andar);
    if (state == GLUT_UP) glutMotionFunc(NULL);
    break;
}
}
break;
case GLUT_RIGHT_BUTTON:
    if (state == GLUT_DOWN) ;
    break;
default:
    break;
}
glutPostRedisplay();
}

```

Se ve que se realizan llamadas a la función callback **glutMotionFunc()** que responde a los movimientos del ratón cuando se tiene pulsado algún botón de este. Cuando se está interactuando con la posición de una luz, se le pasa por ventana una función que realice la operación de acercar o alejar la luz:

```

void Mouse_Luces_Acercar_Alejar(int x, int y) {
    float step;

    step = (float) (y - old_y) / 20.0f;
    old_y = y;
    Acercar_Alejar_Luces( LOCAL_MyLights[current_light], step );
    glutPostRedisplay();
}

```

La función **display()** quedará de la siguiente forma:

```

void display(void) {
    float At[3];
    float Direction[3];

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);

    SetGLCamera( LOCAL_MyCamera );

    SetLight( LOCAL_MyLights[0] );
    SetLight( LOCAL_MyLights[1] );
    SetLight( LOCAL_MyLights[2] );

    glPushMatrix();
    glColor3f(1.0,1.0,0.0);
    drawSphereTurtle();
    switch( current_light ){
    case 0:
    case 1:
    case 2:

```

```

        At[0] = LOCAL_MyLights[current_light]->position[0];
        At[1] = LOCAL_MyLights[current_light]->position[1];
        At[2] = LOCAL_MyLights[current_light]->position[2];
        Direction[0] = - LOCAL_MyLights[current_light]->position[0];
        Direction[1] = - LOCAL_MyLights[current_light]->position[1];
        Direction[2] = - LOCAL_MyLights[current_light]->position[2];
        Draw_Parallel(At);
        Draw_Meridian(At);
        Draw_Vector(At, Direction);
        break;
    default:
        break;
    }
    glPopMatrix();

    glutSwapBuffers();
}

```

Por último, en la función **keyboard()** hay que añadir el siguiente código para que cuando se pulse escape para abandonar el programa, se libere la memoria que se a reservado dinámicamente:

```

case 27:
    DestroyCamera(&LOCAL_MyCamera);
    DestroyLight( LOCAL_MyLights[0] );
    DestroyLight( LOCAL_MyLights[1] );
    DestroyLight( LOCAL_MyLights[2] );
    free (LOCAL_MyLights);
    exit(0);
    break;

```

5.4. TRABAJOS PROPUESTOS

- *Hacer que al acercar o alejar las luces `GL_LIGHT1` o `GL_LIGHT2`, el avance o retroceso de esta mediante el movimiento del ratón sea proporcional a la distancia de dicha luz al centro de la escena.*
- *Proporcionar alguna forma amigable de modificar el color de la luz. Por ejemplo mediante el teclado con las letras (r,g,b para disminuir y R,G,B para aumentar)*

light.h

```
#ifndef LIGHT_H
#define LIGHT_H

#define AGA_DIRECTIONAL 1
#define AGA_POSITIONAL 2
#define AGA_SPOT 3

typedef struct _Light
{
    // The default values that are listed here
    // are those defined by OpenGL
    // Our Default Light has different values
    // (see SetDefaultLight() )

    int type;
    int id; // GL_LIGHTx ; -1 not binded
    int switched; // TRUE => ON
    int needsUpdate; // TRUE / FALSE
    int white; // TRUE / FALSE
    int attenuation; // TRUE / FALSE
    float ambient[4]; // GL_AMBIENT : default (0.0, 0.0, 0.0, 1.0)
    float diffuse[4]; // GL_DIFFUSE : default (0.0, 0.0, 0.0, 1.0)
    // except for light zero (1.0, 1.0, 1.0, 1.0)
    float specular[4]; // GL_SPECULAR : default (0.0, 0.0, 0.0, 1.0)
    // except for light zero (1.0, 1.0, 1.0, 1.0)
    float position[4]; // GL_POSITION : default (0,0,1,0);
    // directional, in the direction of the -z
    float pointAtInfinity[4]; // these values do not refer to the components of
    // one vector they refer to :
    // the coordinates of one point placed in the infinite
    // ( as the point is in the infinite,
    // its 4th homogeneous coordinate must be 0.0 )
    float spotDirection[4]; // GL_SPOT_DIRECTION : default direction is (0,0,-1)
    // significant only when GL_SPOT_CUTOFF is not 180
    float spotExponent; // GL_SPOT_EXPONENT [0,128], default 0
    // 0 => uniform light distribution
    // higher spot => more focused light source,
    float spotCutOff; // GL_SPOT_CUTOFF [0,90] & 180, default 180
    // 180 => uniform light distribution
    float a; // GL_QUADRATIC_ATTENUATION
    float b; // GL_LINEAR_ATTENUATION
    float c; // GL_CONSTANT_ATTENUATION
    //  $I = I / ( a*\delta*\delta + b*\delta + c )$ 
    // delta : distance from light position to point
    // default a=0 b=0 c=1, no atenuation
} light;

light *CreateDefaultLight();
void DestroyLight( light *thisLight );

void SetLight( light *thisLight );
void SetDefaultLight( light *thisLight );
void SwitchLight( light *thisLight, int status );

void Rotar_Luces_Longitud( light *thisLight, float inc );
void Rotar_Luces_Latitud( light *thisLight, float inc );
void Acercar_Alejar_Luces( light *thisLight, float step );

void Rotar_Spot_Latitud( light *thisLight, float inc );
void Rotar_Spot_Longitud( light *thisLight, float inc );

#endif /* LIGHT_H */
```

light.c

```
#include <GL/glut.h>
#include "light.h"
#include "vector_tools.h"

light *CreateDefaultLight() {
    light *newLight;
    newLight = (light *) malloc( sizeof(light) * 1 );

    SetDefaultLight( newLight );
    return newLight;
}

void SetDefaultLight( light *thisLight ) {
    int    ierr = 0;
    float  intensity;
    float  vx, vy, vz;

    // directional light
    thisLight->type      = AGA_DIRECTIONAL;
    thisLight->id        = -1;
    thisLight->switched  = FALSE;
    thisLight->white      = TRUE;
    thisLight->attenuation = FALSE;
    thisLight->needsUpdate = TRUE;

    intensity = 0.0f;
    thisLight->ambient[0] = intensity;
    thisLight->ambient[1] = intensity;
    thisLight->ambient[2] = intensity;
    thisLight->ambient[3] = 1.0f;

    intensity = 0.8f;
    thisLight->diffuse[0] = intensity;
    thisLight->diffuse[1] = intensity;
    thisLight->diffuse[2] = intensity;
    thisLight->diffuse[3] = 1.0f;

    intensity = 0.0f;
    thisLight->specular[0] = intensity;
    thisLight->specular[1] = intensity;
    thisLight->specular[2] = intensity;
    thisLight->specular[3] = 1.0f;

    thisLight->position[0] = 1.0f;
    thisLight->position[1] = 1.0f;
    thisLight->position[2] = 1.0f;
    thisLight->position[3] = 1.0f;

    vx = 1.0f; vy = 1.0f; vz = 1.0f;
    VectorNormalize( &ierr, &vx, &vy, &vz );
    thisLight->pointAtInfinity[0] = vx;
    thisLight->pointAtInfinity[1] = vy;
    thisLight->pointAtInfinity[2] = vz;    // The light is "placed" at point "v" in the infinite
    thisLight->pointAtInfinity[3] = 0.0f;  // So light rays flow in the direction of vector "-v"

    vx = -1.0f; vy = -1.0f; vz = -1.0f;
    VectorNormalize( &ierr, &vx, &vy, &vz );
    thisLight->spotDirection[0] = vx;
    thisLight->spotDirection[1] = vy;
    thisLight->spotDirection[2] = vz;
    thisLight->spotDirection[3] = 0.0f;

    thisLight->spotExponent = 10.0f;
    thisLight->spotCutOff = 30.0f; // must be degrees

    thisLight->a = 0.1f;    // GL_QUADRATIC_ATTENUATION
    thisLight->b = 0.0f;    // GL_LINEAR_ATTENUATION
    thisLight->c = 1.0f;    // GL_CONSTANT_ATTENUATION
}

void DestroyLight( light *thisLight ) {
    if( ! thisLight ) return;
    free( thisLight );
    thisLight = NULL;
}
}
```

light.c

```
void SwitchLight( light *thisLight, int status ) {
    if( ! thisLight ) return;
    if( thisLight->id < GL_LIGHT0 ) return;

    thisLight->switched = status;
    if( status ) {
        glEnable( thisLight->id );
        thisLight->needsUpdate = TRUE;
    }
    else {
        glDisable( thisLight->id );
    }
}

void SetLight( light *thisLight ) {
    int    lightId;

    if( ! thisLight ) return;
    if( ! thisLight->switched ) return;
    if( thisLight->id < GL_LIGHT0 ) return;

    lightId = thisLight->id;

    // Geometric parameters will be always set when the scene is redrawn
    if( thisLight->type == AGA_DIRECTIONAL ) {
        glLightfv( lightId, GL_POSITION, thisLight->pointAtInfinity );
    }
    else if( thisLight->type == AGA_POSITIONAL ) {
        glLightfv( lightId, GL_POSITION, thisLight->position );
    }
    else {
        glLightfv( lightId, GL_POSITION,    thisLight->position );
        glLightfv( lightId, GL_SPOT_DIRECTION, thisLight->spotDirection );
    }

    // These other parameters are seldom changed
    // So, they will be set only when any one of them is changed. The user interface
    // must set "needsUpdate" to TRUE, whenever any of these parameters changes
    if( thisLight->needsUpdate ) {
        thisLight->needsUpdate = FALSE;
        glLightfv( lightId, GL_AMBIENT,    thisLight->ambient );
        glLightfv( lightId, GL_DIFFUSE,    thisLight->diffuse );
        glLightfv( lightId, GL_SPECULAR,   thisLight->specular );
        if( thisLight->type == AGA_SPOT ) {
            glLightf( lightId, GL_SPOT_EXPONENT,    thisLight->spotExponent );
            glLightf( lightId, GL_SPOT_CUTOFF,      thisLight->spotCutOff );
        }
        else {
            glLighti( lightId, GL_SPOT_EXPONENT,    0 );
            glLighti( lightId, GL_SPOT_CUTOFF,      180 );
        }
        if( ! thisLight->attenuation || thisLight->type == AGA_DIRECTIONAL ) {
            glLighti( lightId, GL_CONSTANT_ATTENUATION, 1 );
            glLighti( lightId, GL_LINEAR_ATTENUATION, 0 );
            glLighti( lightId, GL_QUADRATIC_ATTENUATION, 0 );
        }
        else {
            glLightf( lightId, GL_CONSTANT_ATTENUATION,    thisLight->c );
            glLightf( lightId, GL_LINEAR_ATTENUATION,     thisLight->b );
            glLightf( lightId, GL_QUADRATIC_ATTENUATION,  thisLight->a );
        }
    }
}

void Rotar_Luces_Longitud( light *thisLight, float inc ) {
    float vIn[3];

    vIn[0]= thisLight->position[0] ;
    vIn[1]= thisLight->position[1] ;
    vIn[2]= thisLight->position[2] ;

    VectorRotY( vIn, inc );

    thisLight->position[0] = vIn[0];
    thisLight->position[2] = vIn[2];
}
```

light.c

```
void Rotar_Luces_Latitud( light *thisLight, float inc ) {
    float vIn[3];

    vIn[0]= thisLight->position[0] ;
    vIn[1]= thisLight->position[1] ;
    vIn[2]= thisLight->position[2] ;

    VectorRotXZ( vIn, inc, TRUE );

    thisLight->position[0] = vIn[0];
    thisLight->position[1] = vIn[1];
    thisLight->position[2] = vIn[2];
}

void Acercar_Alejar_Luces( light *thisLight, float step ) {
    int ierr;
    float vaX, vaY, vaZ;
    float modulo;

    vaX= thisLight->position[0];
    vaY= thisLight->position[1];
    vaZ= thisLight->position[2];

    VectorNormalize( &ierr, &vaX, &vaY, &vaZ );
    vaX= step * vaX;
    vaY= step * vaY;
    vaZ= step * vaZ;

    // Set new position
    modulo = sqrt(pow(thisLight->position[0] + vaX,2) + pow(thisLight->position[1] + vaY,2) +
        pow(thisLight->position[2] + vaZ,2));
    if(modulo < 0.8f) return;
    thisLight->position[0] += vaX;
    thisLight->position[1] += vaY;
    thisLight->position[2] += vaZ;
}

void Rotar_Spot_Latitud( light *thisLight, float inc ) {
    float vIn[3];

    vIn[0]= thisLight->spotDirection[0] ;
    vIn[1]= thisLight->spotDirection[1] ;
    vIn[2]= thisLight->spotDirection[2] ;

    VectorRotXZ( vIn, inc, TRUE );

    thisLight->spotDirection[0] = vIn[0];
    thisLight->spotDirection[1] = vIn[1];
    thisLight->spotDirection[2] = vIn[2];
}

void Rotar_Spot_Longitud( light *thisLight, float inc ) {
    float vIn[3];

    vIn[0]= thisLight->spotDirection[0] ;
    vIn[1]= thisLight->spotDirection[1] ;
    vIn[2]= thisLight->spotDirection[2] ;

    VectorRotY( vIn, inc );

    thisLight->spotDirection[0] = vIn[0];
    thisLight->spotDirection[2] = vIn[2];
}
```

primitivas.c

```
#include <GL/glut.h>
#include "vector_tools.h"

void Draw_Parallel (float *At) {
    double radius, angle;
    float vectorX,vectorZ, vectorX1, vectorZ1;
    int lightingFlag;

    lightingFlag = glIsEnabled( GL_LIGHTING );
    if( lightingFlag ) glDisable( GL_LIGHTING );
    radius = sqrt(At[0]*At[0]+At[2]*At[2]);
    vectorZ1=radius;
    vectorX1=0.0;
    glBegin(GL_LINE_STRIP);
    for(angle=0.0f;angle<=(2.0f*3.14159);angle+=0.01f){
        vectorX=radius*(float)sin((double)angle);
        vectorZ=radius*(float)cos((double)angle);
        glVertex3d(vectorX1,At[1],vectorZ1);
        vectorZ1=vectorZ;
        vectorX1=vectorX;
    }
    glEnd();
}

void Draw_Meridian (float *At) {
    double radius, alfa, beta;
    float vectorX, vectorY, vectorZ, vectorX1, vectorY1, vectorZ1;
    int lightingFlag;

    lightingFlag = glIsEnabled( GL_LIGHTING );
    if( lightingFlag ) glDisable( GL_LIGHTING );
    radius = sqrt(pow(At[0],2)+pow(At[1],2)+pow(At[2],2));
    alfa = atan2(At[2],At[0]);
    vectorX1=radius*(float)cos((double)alfa);
    vectorY1=0;
    vectorZ1=radius*(float)sin((double)alfa);
    glBegin(GL_LINE_STRIP);
    for(beta=0.0f;beta<=(2.0f*3.14159);beta+=0.01f){
        vectorX=radius*(float)cos((double)beta)*(float)cos((double)alfa);
        vectorY=radius*(float)sin((double)beta);
        vectorZ=radius*(float)cos((double)beta)*(float)sin((double)alfa);
        glVertex3d(vectorX1,vectorY1,vectorZ1);
        vectorX1=vectorX;
        vectorY1=vectorY;
        vectorZ1=vectorZ;
    }
    glEnd();
}

void Draw_Vector(float *At, float *Direction) {
    int ierr, lightingFlag;
    float mod;
    float alpha,beta;
    float length = .2f;
    float vectorX, vectorY, vectorZ;

    lightingFlag = glIsEnabled( GL_LIGHTING );
    if( lightingFlag ) glDisable( GL_LIGHTING );
    mod = sqrt(pow(Direction[0],2)+pow(Direction[1],2)+pow(Direction[2],2));
    alpha = atan2(Direction[0],Direction[2]);
    beta = asin(Direction[1]/mod);
    glBegin(GL_LINES);
    glColor3f(1.0f,0.0f,0.0f);
    glVertex3f(At[0], At[1], At[2] );
    glVertex3f(At[0]+Direction[0]*length,At[1]+Direction[1]*length,At[2]+Direction[2]*length);
    glEnd();
    VectorNormalize( &ierr, &Direction[0], &Direction[1], &Direction[2]);
    vectorX = At[0] + Direction[0]*(length-0.05);
    vectorY = At[1] + Direction[1]*(length-0.05);
    vectorZ = At[2] + Direction[2]*(length-0.05);
    glTranslatef(vectorX, vectorY, vectorZ);
    glRotatef(beta *RAD_TO_DEGREE, sin( alpha-PI_VALUE/2.0 ), 0.0f, cos( alpha-PI_VALUE/2.0 ));
    glRotatef( alpha*RAD_TO_DEGREE, 0.0f, 1.0f, 0.0f );
    glutSolidCone(0.02,0.1,28,28);
}
}
```

primitivas.c

```
void Draw_Sphere_Spot(float *At, float *Direction) {
    int lightingFlag;
    float mod;
    float alpha,beta, alfa;
    float length = .2f;
    double radius, angle;
    float vectorX, vectorY, vectorZ, vectorX1, vectorY1, vectorZ1;

    lightingFlag = glIsEnabled( GL_LIGHTING );
    if( lightingFlag ) glDisable( GL_LIGHTING );
    mod = sqrt(pow(Direction[0],2)+pow(Direction[1],2)+pow(Direction[2],2));
    alpha = atan2(Direction[0],Direction[2]);
    beta = asin(Direction[1]/mod);
    glLoadIdentity();
    radius = sqrt(pow(Direction[0]*length,2)+pow(Direction[2]*length,2));
    vectorX1=At[0];
    vectorZ1=At[2]+radius;
    glBegin(GL_LINE_STRIP);
    for(angle=0.0f;angle<=(2.0f*3.14159);angle+=0.01f){
        vectorX=At[0]+radius*(float)sin((double)angle);
        vectorZ=At[2]+radius*(float)cos((double)angle);
        glVertex3d(vectorX1,At[1]+Direction[1]*length,vectorZ1);
        vectorZ1=vectorZ;
        vectorX1=vectorX;
    }
    glEnd();
    radius = sqrt( pow(Direction[0]*length,2) + pow(Direction[1]*length,2) +
        pow(Direction[2]*length,2) );
    alfa = atan(Direction[2]/Direction[0]);
    vectorX1=At[0]+radius*(float)cos((double)alfa);
    vectorY1=At[1];
    vectorZ1=At[2]+radius*(float)sin((double)alfa);
    glBegin(GL_LINE_STRIP);
    for(beta=0.0f;beta<=(2.0f*3.14159);beta+=0.01f){
        vectorX=At[0]+radius*(float)cos((double)beta)*(float)cos((double)alfa);
        vectorY=At[1]+radius*(float)sin((double)beta);
        vectorZ=At[2]+radius*(float)cos((double)beta)*(float)sin((double)alfa);
        glVertex3d(vectorX1,vectorY1,vectorZ1);
        vectorX1=vectorX;
        vectorY1=vectorY;
        vectorZ1=vectorZ;
    }
    glEnd();
}

void drawSphereTurtle() {
    int slices = 40;
    int stacks = 40;

    glPushMatrix();
        glScalef(1.0f,.3f,1.0f);
        glutSolidSphere(1.0,slices,stacks);
    glPopMatrix();
    glPushMatrix();
        glTranslatef(.7f,0.0f,.7f);
        glutSolidSphere(.3,slices,stacks);
    glPopMatrix();
    glPushMatrix();
        glTranslatef(-.7f,0.0f,.7f);
        glutSolidSphere(.3,slices,stacks);
    glPopMatrix();
    glPushMatrix();
        glTranslatef(.7f,0.0f,-.7f);
        glutSolidSphere(.3,slices,stacks);
    glPopMatrix();
    glPushMatrix();
        glTranslatef(-.7f,0.0f,-.7f);
        glutSolidSphere(.3,slices,stacks);
    glPopMatrix();
    glPushMatrix();
        glScalef(1.0f,.6f,1.0f);
        glTranslatef(0.0f,0.0f,-1.2f);
        glutSolidSphere(.4,slices,stacks);
    glPopMatrix();
}
```