PRÁCTICA 4
MATERIALES, FUENTES DE LUZ Y TEXTURAS

ALBERTO ORTEGA VILCHEZ

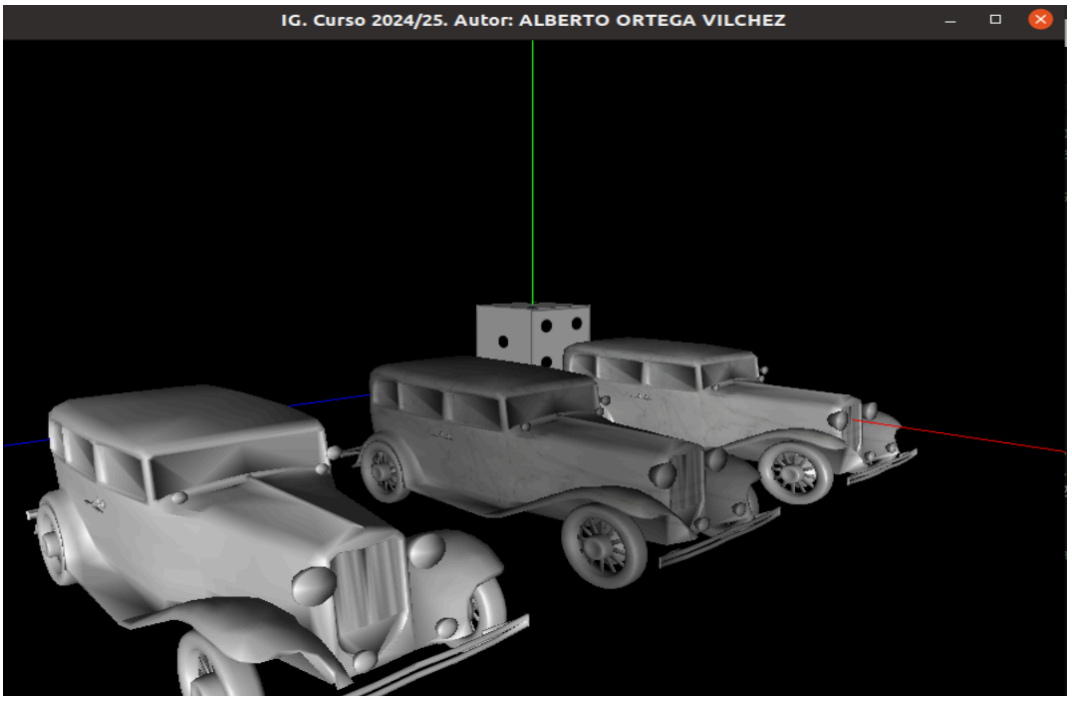
1 - Índice

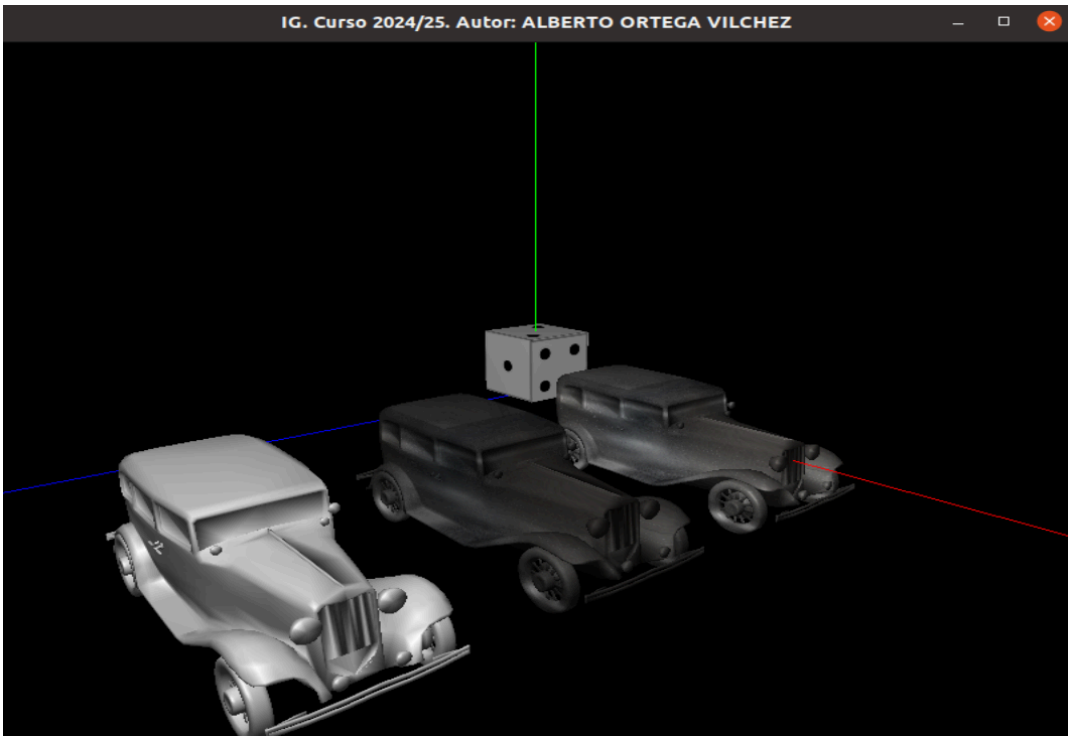
2 - Introducción

Utilizaré este documento como memoria de la práctica 4 en la que trabajo con iluminación y texturas.

Describiré de manera resumida cómo he abordado cada uno de los apartados solicitados en esta práctica.

Utilizaré como código base el utilizado en la práctica 2, pues es donde desarrollé la clase Malla y esta será la que sufrirá más modificaciones.





3 - Desarrollo de la práctica

Durante el desarrollo de la práctica he diferenciado 2 partes. La primera parte, que corresponde a la asignación manual de coordenadas de textura específicas al Dado, y la segunda, en la que se realizan las funciones de asignación de coordenadas

3.1 - Añadir materiales y textura a las mallas

Defino los siguientes atributos en la clase Malla.h:

```
// 1) Añadir Materiales y textura a las mallas

/**
 * @brief Reflectividad difusa del material en formato RGBA
*/
GLfloat reflectividad_difusa[4] = { 0.8f, 0.8f, 0.8f, 1.0f };

/**
 * @brief Reflectividad especular del material en formato RGBA
*/
GLfloat reflectividad_especular[4] = { 0.0f, 0.0f, 0.0f, 1.0f };

/**
 * @brief Reflectividad ambiente del material en formato RGBA
*/
GLfloat reflectividad_ambiente[4] = { 0.2f, 0.2f, 0.2f, 1.0f };

/**
 * @brief Exponente especular que define el brillo del material
*/
float e=0.0;
```

Defino los métodos para asignar estos atributos en mis mallas:

```
/**
 * @brief Asigna la reflectividad difusa del material
 * @param r componente roja
 * @param g componente verde
 * @param b componente azul
 * @param alfa componente alfa. Valor predeterminado 1.0f
*/
void asignarReflectividadDifusa(GLfloat r, GLfloat g, GLfloat b, GLfloat alfa=1.0f);

/**
 * @brief Asigna la reflectividad especular del material
 * @param r componente roja
 * @param g componente verde
 * @param b componente azul
 * @param alfa componente alfa. Valor predeterminado 1.0f
*/
void asignarReflectividadEspecular(GLfloat r, GLfloat g, GLfloat b, GLfloat alfa=1.0f);

/**
 * @brief Asigna la reflectividad ambiente del material
 * @param r componente roja
 * @param g componente verde
 * @param b componente azul
 * @param alfa componente alfa. Valor predeterminado 1.0f
*/
void asignarReflectividadAmbiente(GLfloat r, GLfloat g, GLfloat b, GLfloat alfa=1.0f);
```

```
/**
 * @brief Asigna el exponente especular del material
 * @param exp Exponente especular. Valor predeterminado 0.0f
*/
void asignarExponenteEspecular(float exp=0.0f);
```

Implemento los métodos mencionados:

```
void Malla::asignarReflectividadDifusa(GLfloat r, GLfloat g, GLfloat b, GLfloat alfa){
    reflectividad_difusa[0]=r;
    reflectividad_difusa[1]=g;
    reflectividad_difusa[2]=b;
    reflectividad_difusa[3]=alfa;

    // Hacemos que la reflectividad ambiente sea igual que la difusa
    // a menos de que la reflectividad_ambiente sea establecida, en
    // cuyo caso se sobreescribirán estos valores.
    reflectividad_ambiente[0]=reflectividad_difusa[0];
    reflectividad_ambiente[1]=reflectividad_difusa[1];
    reflectividad_ambiente[2]=reflectividad_difusa[2];
    reflectividad_ambiente[3]=reflectividad_difusa[3];
}

void Malla::asignarReflectividadEspecular(GLfloat r, GLfloat g, GLfloat b, GLfloat alfa){
    reflectividad_especular[0]=r;
    reflectividad_especular[1]=g;
    reflectividad_especular[2]=b;
    reflectividad_especular[3]=alfa;
}

void Malla::asignarReflectividadAmbiente(GLfloat r, GLfloat g, GLfloat b, GLfloat alfa){
    reflectividad_ambiente[0]=r;
    reflectividad_ambiente[1]=g;
    reflectividad_ambiente[2]=b;
    reflectividad_ambiente[3]=alfa;
}


void Malla::asignarExponenteEspecular(float exp){
    e=exp;
}
```

3.1.1 - Configuraciones predeterminadas

Para agilizar la asignación de materiales a las mallas, de forma que se puedan percibir las distintas reflectividades, he establecido 3 configuraciones predeterminadas que harán que se aprecie bien la diferencia.

```
void Malla::configuracionMaterial1(){
    asignarReflectividadAmbiente(0.25f,0.25f,0.25f,1.0f);
    asignarReflectividadDifusa(0.6f,0.6f,0.6f,1.0f);
    asignarReflectividadEspecular(0.9f,0.9f,0.9f,1.0f);
    asignarExponenteEspecular(100.0f);
}

void Malla::configuracionMaterial2(){
    asignarReflectividadAmbiente(0.2f,0.2f,0.2f,1.0f);
    asignarReflectividadDifusa(0.4f,0.4f,0.4f,1.0f);
    asignarReflectividadEspecular(0.1f,0.1f,0.1f,0.1f);
    asignarExponenteEspecular(10.0f);
}

void Malla::configuracionMaterial3(){
    asignarReflectividadAmbiente(0.15f,0.15f,0.15f,1.0f);
    asignarReflectividadDifusa(0.7f,0.7f,0.7f,1.0f);
    asignarReflectividadEspecular(0.5f,0.5f,0.5f,0.5f);
    asignarExponenteEspecular(50.0f);
}
```

3.2 - Asignar una textura leída de un archivo a un objeto

Seguiré el guión de prácticas para implementar el método “cargarTextura”, que leerá un archivo.

```
/**
 * @brief Identificador de la textura asociada a la malla
*/
GLuint texId;
```

```
void Malla::cargarTextura(const char *archivo){
    // Serán unsigned porque son los tipos de datos usados en lector-jpg.cpp
    unsigned width,height;
    unsigned char *imagen = LeerArchivoJPEG(archivo,width,height);

    if(!imagen){
        cout << "\nNo se ha cargado la imagen de textura \n";
        tieneTextura=false;
    }else{
        cout << "\nImagen para textura leida con éxito \n- " << archivo << "\n- Altura: " << height << "\n-Anchura: " << width << "\n\n";
    }

    glGenTextures(1,&texId);
    glBindTexture(GL_TEXTURE_2D, texId);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, imagen);

    delete[] imagen; // Liberamos la memoria usada

    tieneTextura=true;
}
```

3.3 - Textura del Dado - CLASE DADO

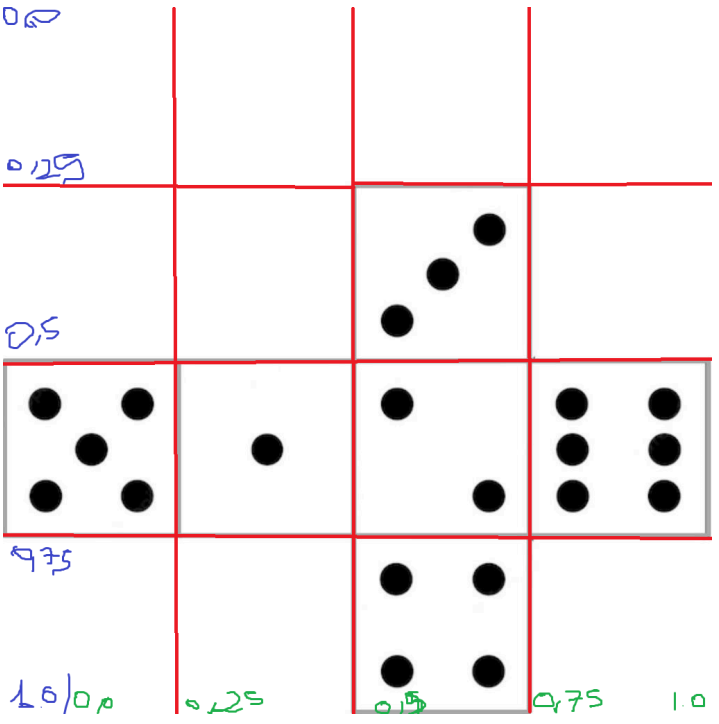
En el caso del dado, he adaptado la antigua clase “Cubo” que tenía para dar lugar a la clase “Dado”. Me valdré de las funcionalidades ya desarrolladas para las mallas; principalmente las relativas a la asignación de reflectividades y el exponente especular.

```
class Dado : public Malla{

    private:
        /**
         * @brief Lado del dado
        */
        float lado;

        /**
         * @brief Mitad del lado del dado
        */
        float mitad;

    public:
        /**
         * @brief Constructor de un dado (cubo) a partir de un tamaño de lado dado
         * @param f Será la longitud del lado
        */
        Dado(float f);

        /**
         * @brief Función "sobrecargada" de la clase Malla donde creo mi dado.
        */
        void draw();
};

#endif // DADO_H
```

De esta forma, la única funcionalidad real a implementar en el dado será la asignación explícita de las coordenadas de textura.

Para ello, simplemente diseccionо la imagen proporcionada determinando las coordenadas de textura exactas (dado que están normalizadas en el intervalo [0,1] y se aprecia que la imagen se puede interpretar como una matriz 4x4.

Asignaré a mi voluntad cada cara del dado en cada cara del cubo



```
22  void Dado::draw() {
23      glMaterialfv(GL_FRONT, GL_AMBIENT, reflectividad_ambiente);
24      glMaterialfv(GL_FRONT, GL_DIFFUSE, reflectividad_difusa);
25      glMaterialfv(GL_FRONT, GL_SPECULAR, reflectividad_especular);
26      glMaterialf(GL_FRONT, GL_SHININESS,e);
27      glEnable(GL_TEXTURE_2D);
28      glBindTexture(GL_TEXTURE_2D, texId);
29
30      glBegin(GL_QUADS);
31      // Cara frontal (Número 1)
32      glNormal3f(0.0f, 0.0f, 1.0f);
33      glTexCoord2f(0.25f, 0.75f); glVertex3f(-mitad, 0.0f, mitad);  // Vértice inferior izquierdo
34      glTexCoord2f(0.5f, 0.75f); glVertex3f(mitad, 0.0f, mitad);   // Vértice inferior derecho
35      glTexCoord2f(0.5f, 0.5f); glVertex3f(mitad, lado, mitad);  // Vértice superior derecho
36      glTexCoord2f(0.25f, 0.5f); glVertex3f(-mitad,  lado, mitad);  // Vértice superior izquierdo
37      // Cara trasera (Número 6)
38      glNormal3f(0.0f,0.0f,-1.0f);
39      glTexCoord2f(0.75f, 0.75f); glVertex3f(mitad, lado, -mitad);
40      glTexCoord2f(1.0f, 0.75f); glVertex3f( mitad, 0.0f, -mitad);
41      glTexCoord2f(1.0f, 0.5f); glVertex3f( -mitad,  0.0f, -mitad);
42      glTexCoord2f(0.75f, 0.5f); glVertex3f(-mitad, lado, -mitad);
43      // Cara izquierda (Número 3)
44      glNormal3f(-1.0f, 0.0f, 0.0f);
45      glTexCoord2f(0.5f, 0.5f); glVertex3f(-mitad, 0.0f, -mitad);
46      glTexCoord2f(0.75f, 0.5f); glVertex3f(-mitad, 0.0f,  mitad);
47      glTexCoord2f(0.75f, 0.25f); glVertex3f(-mitad,  lado,  mitad);
48      glTexCoord2f(0.5f, 0.25f); glVertex3f(-mitad, lado, -mitad);
49      // Cara derecha (Número 4)
50      glNormal3f(1.0f, 0.0f, 0.0f);
51      glTexCoord2f(0.5f, 1.0f); glVertex3f( mitad, 0.0f, -mitad);
52      glTexCoord2f(0.75, 1.0f); glVertex3f( mitad, lado, -mitad);
53      glTexCoord2f(0.75f, 0.75f); glVertex3f( mitad, lado, mitad);
54      glTexCoord2f(0.5f, 0.75f); glVertex3f( mitad,  0.0f, mitad);
55      // Cara superior (Número 2)
56      glNormal3f(0.0f, 1.0f, 0.0f);
57      glTexCoord2f(0.75f, 0.5f); glVertex3f(mitad,  lado, -mitad);
58      glTexCoord2f(0.5f, 0.5f); glVertex3f( -mitad, lado, -mitad);
59      glTexCoord2f(0.5f, 0.75f); glVertex3f( -mitad, lado, mitad);
60      glTexCoord2f(0.75f, 0.75f); glVertex3f(mitad, lado, mitad);
61      // Cara inferior (Número 5)
62      glNormal3f(0.0f, -1.0f, 0.0f);
63      glTexCoord2f(0.25f, 0.5f); glVertex3f(-mitad, 0.0f, -mitad);
64      glTexCoord2f(0.0f, 0.5f); glVertex3f( mitad, 0.0f, -mitad);
65      glTexCoord2f(0.0f, 0.75f); glVertex3f( mitad, 0.0f, mitad);
66      glTexCoord2f(0.25f, 0.75f); glVertex3f(-mitad, 0.0f, mitad);
67      glEnd();
68
69      glDisable(GL_TEXTURE_2D);
70  }
```

3.4 - Añadir coordenadas de textura a las mallas

Utilizaré un vector<pair<float,float>> para almacenar las coordenadas de textura (u,v) y un booleano **tieneTextura** para gestionar si la malla debe procesarse como una malla con textura o una malla sin textura (en cuyo caso, no sería necesario calcular las coordenadas de textura ni procesar ninguna imagen).

```
// 4) Añadir coordenadas de textura a las mallas

/**
 * @brief Vector que almacenará las coordenadas de textura para cada vértice de la malla
*/
vector<pair<float,float>> coordenadasTextura;

/**
 * @brief Será true si la Malla tiene una textura asociada y false en caso contrario
*/
bool tieneTextura=false;
```

3.5 - Calcular las coordenadas de textura a las mallas

Seguiré el apoyo que se encuentra en Prado para la creación de cajas AABB (bounding box) y normalización de coordenadas, adaptado según mi implementación de la clase Malla:

3.5.1 - Calcular el centro

Calculo el punto de referencia desde el que se calculará la orientación de cada vértice en el espacio; es decir, el centro de mi malla

```
Punto3D Malla::calcularCentro(){
    Punto3D centro;

    // Sumo cada coordenada de cada vértice de mi malla y defino el centro haciendo
    // un promedio de cada una de estas coordenadas.
    for(int i=0;i<vertices.size();i++){
        Punto3D v=vertices[i];

        centro.x += v.x;
        centro.y += v.y;
        centro.z += v.z;
    }

    centro.x = centro.x/vertices.size();
    centro.y = centro.y/vertices.size();
    centro.z = centro.z/vertices.size();

    return centro;
}
```

3.5.2 - Calcular la caja

Para normalizar las coordenadas de los vértices de cada eje, de forma que las coordenadas de textura (u,v) estén en el rango [0,1], implemento el siguiente método:

```
void Malla::calcularCajaEnvolvente(Punto3D& min, Punto3D& max){

    if(vertices.empty()){
        cout << "\nERROR CCE -> No hay vertices\n";
        return;
    }

    // Establezco min y max al primer vértice de la malla
    // (podria ser cualquiera en realidad pero facilitira la iteración)
    min=vertices[0];
    max=vertices[0];

    // Recorro todos los vértices para encontrar la coordenada más alta y más baja,
    // que serán asignadas respectivamente a cada coordenada de min y max
    for(int i=1;i<vertices.size();i++){
        if(vertices[i].x < min.x){
            min.x=vertices[i].x;
        }

        if(vertices[i].x > max.x){
            max.x=vertices[i].x;
        }

        if(vertices[i].y < min.y){
            min.y=vertices[i].y;
        }

        if(vertices[i].y > max.y){
            max.y=vertices[i].y;
        }

        if(vertices[i].z < min.z){
            min.z=vertices[i].z;
        }

        if(vertices[i].z > max.z){
            max.z=vertices[i].z;
        }
    }

    // En este momento, min y max representarán las "esquinas" de la caja envolvente
}
```

3.5.3 - Calcular las coordenadas

Para calcular las coordenadas de textura se pueden utilizar diferentes proyecciones tal y como se indica en el guión de prácticas. Me ayudaré de las funciones definidas anteriormente.

Pese a estar implementados las 3 formas, utilizaré la versión del cálculo de coordenadas en textura cilíndrica en mis ejemplos y pruebas.

Plano → Se deberá utilizar en objetos o superficies planos como el suelo o la pared.

```
void Malla::calculoCoordenadasTexturaPlano(){
    Punto3D min,max;
    calcularCajaEnvolvente(min,max);

    float ancho = max.x - min.x;
    float profundidad = max.z - min.z;

    for(int i=0;i<vertices.size();i++){
        float u=(vertices[i].x - min.x)/ancho;
        float v=(vertices[i].z - min.z)/profundidad;

        pair<float,float> ct(u,v);
        coordenadasTextura.push_back(ct);
    }
}
```

Esfera → Se utilizará con mallas que tengan formas esféricas como planetas

```
void Malla::calculoCoordenadasTexturaEsfera(){
    Punto3D centro=calcularCentro();

    Punto3D min,max;

    calcularCajaEnvolvente(min,max);

    float altura=max.y - min.y;
    float radioMax = sqrt((max.x-centro.x) * (max.x-centro.x)) + ((max.y-centro.y)*(max.y-centro.y)) + ((max.z-centro.z)*(max.z-centro.z));

    for(int i=0;i<vertices.size();i++){
        float dx = vertices[i].x - centro.x;
        float dz = vertices[i].z - centro.z;
        float u = 0.5f + (dx/radioMax);

        float dy = vertices[i].y - centro.y;
        float distanciaV = sqrt(dx*dx + dy*dy + dz*dz);
        float v = 0.5f - (dy/distanciaV) * 0.5f;

        pair<float,float> ct(u,v);

        coordenadasTextura.push_back(ct);
    }
}
```

Cilíndrica → Se recomienda su uso en objetos con simetría cilíndrica como columnas

```
void Malla::calculoCoordenadasTexturaCilindrica(){
    coordenadasTextura.clear();

    Punto3D centro=calcularCentro(); // Calculamos el centro geométrico

    Punto3D min,max;
    // Calculamos la caja envolvente para definir los límites del obejto en cada eje, facilitando
    // la normalización de las coordenadas en el eje Y
    calcularCajaEnvolvente(min,max);

    float altura=max.y - min.y;

    for(size_t i=0; i<vertices.size(); ++i){

        // Calculo la distancia de las coordenadas X y Z de cada vértice respecto al centro calculado
        float dx = vertices[i].x - centro.x;
        float dz = vertices[i].z - centro.z;

        float distanciaXZ = sqrt(dx*dx + dz*dz);

        // Coordenada u -> Representa un ángulo aproximado en todno al eje Y de la malla
        float u = 0.5f + (dx/distanciaXZ) * 0.5f;

        // Coordenada v -> Se obtiene a partir de la posición Y de cada vértice, normalizada
        // dentro de la altura de la caja envolvente. Esto permite distribuir "v" en [0,1]
        float v = (vertices[i].y - min.y)/altura;

        // Almacenamos u,v como un par de valores
        pair<float,float> ct(u,v);
        coordenadasTextura.push_back(ct);
    }
}
```

3.6 - Dibujo con textura

En mi caso, utilizo dos funciones distintas para dibujar, dependiendo del sombreado asignado a la malla.

De la siguiente forma gestiono que el dibujo de las mallas utilice textura siempre y cuando se le asigne una:

```
void Malla::cargarTextura(const char *archivo){
    // Serán unsigned porque son los tipos de datos usados en lector-jpg.cpp
    unsigned width,height;
    unsigned char *imagen = LeerArchivoJPEG(archivo,width,height);

    if(!imagen){
        cout << "\nNo se ha cargado la imagen de textura \n";
        tieneTextura=false;
    }else{
        cout << "\nImagen para textura leida con éxito \n- " << archivo << "\n- Alt
    }

    glGenTextures(1,&texId);
    glBindTexture(GL_TEXTURE_2D, texId);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BY

    delete[] imagen; // Liberamos la memoria usada

    tieneTextura=true;
}
```

Tanto en el caso de **drawFlat** como en **drawSmooth** hago una comprobación previa a cada dibujo de cada vértice. Muestro un ejemplo que se repite con cada vértice:

```
// Práctica 4 - Asigno las coordenadas de textura
if(tieneTextura){glTexCoord2f(coordenadasTextura[t.getI0()].first, coordenadasTextura[t.getI0()].second);};
glVertex3f(this->vertices[t.getI0()].x , this->vertices[t.getI0()].y , this->vertices[t.getI0()].z);
```

3.7 - Creación de la escena

Mi escena contendrá un Dado y 3 coches.

```
// PRÁCTICA 4 - Mallas a dibujar y Dado

Dado dado(4.0f); // Dado "hereda" de Malla
Malla coche1("plys/big_dodge.ply",false); // Reflectividad difusa
Malla coche2("plys/big_dodge.ply",false); // Reflectividad ambiente
Malla coche3("plys/big_dodge.ply",false); // Reflectividad especular
```

Asignaré los siguientes materiales

```
void
initModel (){
  // Práctica 4
  dado.cargarTextura("JPEG/dado.jpg");
  dado.asignarReflectividadDifusa(0.5f,0.5f,0.5f,1.0f);
  dado.asignarReflectividadEspecular(0.8f,0.8f,0.8f,1.0f);
  dado.asignarReflectividadAmbiente(1.0f,1.0f,1.0f,1.0f);
  dado.asignarExponenteEspecular(100.0f);
  dado.setSombreadoSuave(true);

  coche1.cargarTextura("JPEG/texturaMarmol.jpg");
  coche1.calculoCoordenadasTexturaCilindrica();
  coche1.configuracionMaterial1();
  coche1.setSombreadoSuave(true);

  coche2.cargarTextura("JPEG/texturaMarmol.jpg");
  coche2.calculoCoordenadasTexturaCilindrica();
  coche2.configuracionMaterial2();
  coche2.setSombreadoSuave(true);

  coche3.configuracionMaterial3();
  coche3.setSombreadoSuave(true);
}
```

Dibujo y distribuyo los objetos en la escena de la siguiente forma en el método Dibuja():

```
 dado.draw();

 glTranslatef(10.0,0.0,-5.0);
 coche1.draw();

 glTranslatef(0.0,0.0,10.0);
 coche2.draw();

 glTranslatef(0.0,0.0,10.0);
 coche3.draw();
```

4 - Uso del teclado

TECLAS W / **w** → Alterno entre los dos focos de luz definidos:

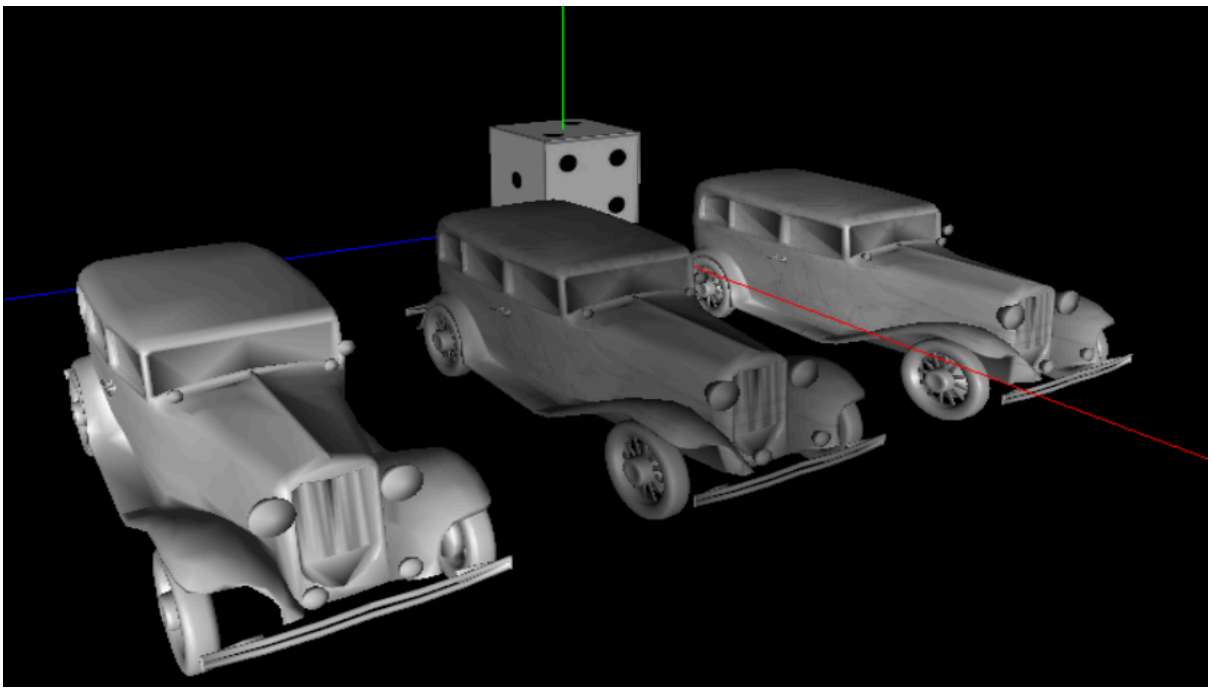
Cada vez que se pulsan, se llama al siguiente método.

```
void establecerLuzActiva(){

  luzActiva = !luzActiva;

  if(luzActiva){
    glLightfv(GL_LIGHT0,GL_POSITION,posLuz1);
  }else{
    glLightfv(GL_LIGHT0,GL_POSITION,posLuz2);
  }
}
```

Las dos posiciones de luz son las siguientes:

```
// PRÁCTICA 4 - Dos focos de luz distintos

GLfloat posLuz1[4] = {5.0,5.0,10.0,0.0}; // Será el foco de luz por defecto (la que hemos tenido hasta ahora)
GLfloat posLuz2[4] = {-5.0,10.0,-5.0,1.0};

bool luzActiva=true;
```

POSICIÓN 1:



POSICIÓN 2: