TUTORIAL #4: Iluminación y texturas

El contenido de este tutorial es explicar y entender como OpenGL trabaja con el sistema de iluminación, el cual es una pieza fundamental en el color de los objetos; Y del sistema de texturas, que nos permite colocar imágenes en superficies para crear efectos de realismo que no serian posibles con las técnicas ya vistas en clases anteriores.

Iluminación

La iluminación es un elemento fundamental en la computación grafica, ya que nos permite dar realismo a las escenas que estamos haciendo. OpenGL puede manejar hasta 8 luces con sus diversas propiedades simultáneamente. Primero que nada para prender la iluminación aplicamos el comando **glEnable(GL_LIGHTING);** y para prender cada luz individualmente aplicamos **glEnable(GL_LIGHT0);** donde 0 puede ser cualquiera de las 8 luces que se desea prender(del 0 al 7). Por defecto OpenGL tiene desactivadas todas las luces.

Luces y colores:

Primero veamos como asignar una luz ambiental Global, la cual afectara a todos los objetos de la escena:

```
Glfloat global_ambient[] = { 0.5f, 0.5f, 0.5f, 1.0f };

glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient);
```

Esto nos da un ambiente grisáceo cuando tengamos la iluminación prendida pero ninguna luz activada.

Ahora veamos que propiedades podemos agregar a nuestras luces:

Ambiental: Este valor indica cuanta luz ambiental se debe agregar al sistema cuando está encendida, no hay que confundirla con la luz ambiental global ya que esta solo afecta cuando esta luz está encendida, en cambio la global afecta aunque todas las luces estén apagadas.
Difusa: Es el valor que indica que color de luz estamos mostrando de manera direccional, esta luz está afectada por el vector que se está usando(a diferencia de la luz ambiental)
Especular: Igual que la luz difusa, esta es direccional, pero la luz especular muestra el valor de luz que se refleja al golpear la superficie y está afectada por el brillo de la superficie que golpee (más información adelante en Materiales).
Posición: Es un vector que indica que dirección tiene la luz, este vector solo aplica para la luz Difusa y Especular, ya que la ambiental afecta toda la escena.

Para colocarle estas propiedades a alguna luz (nuestro ejemplo a GL_LIGHT0) seguimos estas instrucciones:

```
//Creamos una luz roja un poco a la derecha de la esfera

const GLfloat light_ambient[]  = { 0.0f, 0.0f, 0.0f, 1.0f };

const GLfloat light_diffuse[]  = { 1.0f, 0.0f, 0.0f, 1.0f };

const GLfloat light_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };

const GLfloat light_position[] = { 5.0f, 0.0f, 5.0f, 0.0f };

//Asignamos estos valores a la luz

 glLightfv(GL_LIGHT0, GL_AMBIENT,  light_ambient);

   glLightfv(GL_LIGHT0, GL_DIFFUSE,  light_diffuse);

   glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);

   glLightfv(GL_LIGHT0, GL_POSITION, light_position);

//Activamos la luz 0

glEnable(GL_LIGHT0);
```

Colores y Materiales:

Cuando activamos el sistema de luces, los colores asignados por el glColor no se comportan correctamente. Al tener una luz activa, solo se verá el color de esta. Si queremos que nuestros objetos tengan colores particulares, debemos utilizar una nueva instrucción llamada **glEnable(GL_COLOR_MATERIAL);** la cual permite que los colores especificados interactúen con las luces. Generalizando esto, cuando una luz esta prendida, se deben asignar cualidades al material (objeto) para que este se pueda ver en la luz, esto es, características que quieres que el objeto posea:

Ambiental: Es el color que muestra el objeto cuando es afectado por luz ambiental.
Difusa: Color que posee el objeto al estar enfrente de un haz de luz.
Especular: es el color que refleja el objeto cuando se le aplica luz
Brillo: es un coeficiente que indica cuanto brillo refleja el objeto.

Cuando asignamos las propiedades del material, todas las operaciones siguientes serán afectadas por este cambio, así que debemos asignar estos componentes por cada objeto que se desplegara antes de dibujarlos.

Veamoslo con un ejemplo:

```
//Definimos un material

const GLfloat mat_ambient[]    = { 0.0f, 0.0f, 0.0f, 1.0f };

const GLfloat mat_diffuse[]    = { 0.0f, 1.0f, 0.0f, 1.0f };

const GLfloat mat_specular[]   = { 1.0f, 1.0f, 1.0f, 1.0f };

const GLfloat high_shininess[] = { 10.0f };

//Asignamos estos valores a los componentes del material

   glMaterialfv(GL_FRONT, GL_AMBIENT,   mat_ambient);

   glMaterialfv(GL_FRONT, GL_DIFFUSE,   mat_diffuse);

   glMaterialfv(GL_FRONT, GL_SPECULAR,  mat_specular);

   glMaterialfv(GL_FRONT, GL_SHININESS, high_shininess);
```

Algo que hay que recalcar es que el color final de un objeto viene después de operar los colores de las luces con los colores del material, si aplicamos el código anterior de la luz roja y le agregamos este código de un material, veremos solo la parte especular de la iluminación, ya que el componente ambiental de ambos es 0 y en el componente difuso los colores no corresponden, es una luz roja en un objeto verde, así que no se verá el objeto, solo se verá su brillo (especular) que será una luz blanca (ya que tanto en la luz como en el material el componente especular esta en (1,1,1)).

Texturas

En muchas oportunidades, asignar colores y materiales no es suficiente para crear el realismo necesario, si deseamos dibujar un avión tendríamos que cambiar colores y por cada zona del avión y además, tendríamos un objeto demasiado complejo para representar todo el avión, una solución a este problema son las texturas. Las texturas son imágenes de mapas de bits que se colocan en un polígono, como si forráramos un cuaderno con papel adhesivo, el cual envolverá toda la cara del cuaderno.

Nombre de las texturas:

Las texturas deben tener un identificador, que internamente para OpenGL es un número (pero no nos preocupemos por eso). Para asignar un nombre a nuestra textura usaremos:

```
GLuint textura;
```

```
glGenTexture(1,&textura);

//Para utilizart la textura utilizamos

glBindTexture(GL_TEXTURE_2D,textura);
```

Luego de que leamos la información de la textura (leer el archivo y convertirlo en bytes, por lo que se recomienda utilizar archivos BMP, ya que son los más sencillos de leer) le decimos a OpenGl que cree una textura a partir de esos datos:

```
/*El primer parámetro indica que es una textura 2D,luego viene el nivel de detalle de la textura, dejémosla en 0 por ahora, el 3er parámetro son las componentes que posee la data(RGB),luego es el borde, que usualmente es 0, GL_RGB le dice a OpenGL que los datos que usamos están en ese orden RGB, GL_UNSIGNED_BYTE es el tipo de dato que estamos usando en la textura y finalmente "textura" es de donde estamos tomando toda esta información*/

glTexImage2D(GL_TEXTURE_2D, 0, 3, Width, ImageHeight, 0, GL_RGB,
GL_UNSIGNED_BYTE,textura);
```

Ya tenemos nuestra textura cargada y lista, ahora le diremos a OpenGL algunos parámetros de cómo debe usar nuestra textura:

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,GL_NEAREST);

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,GL_NEAREST);
```

En el cuadro anterior le dijimos a OpenGL que las texturas 2D se aproximen al polígono cuando la textura es muy grande o muy pequeña ,esto hará que la textura se vea un poco "cuadriculada" si queremos ver más natural este ajuste cambiamos la instrucción GL_NEAREST a GL_LINEAR, aunque GL_LINEAR requiere más computo.

Ahora para utilizar nuestra textura ya cargada solo colocamos antes de cara vértice glTexCoord2f(x,y); donde 'x' e'y' son las coordenadas de la textura que serán asignados al próximo glVertex que haya, esto es básicamente decir de donde a donde usaremos la textura. Estos valores van de 0.0 a 1.0, pero en algunos casos puede que necesitemos ir más allá de estos valores. Para saber el comportamiento de OpenGL debemos asignar este parámetro:

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
```

Si colocamos GL_CLAMP lo que haremos será convertir las coordenadas de textura y adaptarlas a la textura, es decir, valores mayores que 1.0 se volverán 1.0 y valores menores de 0.0 se volverán 0.0.

Otra opción es en vez de colocar GL_CLAMP es usar GL_REPEAT que hace es repetir la textura como sea necesario, así si colocamos una coordenada mayor OpenGL repetirá la textura lo necesario para que exista esa coordenada de textura y utilizarla

Referencias

OpenGL @ Lighthouse 3D - http://www.lighthouse3d.com/OpenGL/glut/index.php

NeHe Productions - http://nehe.gamedev.net

OpenGL Texture Tutorial: http://www.nullterminator.net/gltexture.html

The Red Book - OpenGL Programming Guide (Addison-Wesley Publishing Company):