Programación 3D

Práctica 5: Iluminación

En esta práctica, añadiremos fuentes de luz al motor. De las vistas en la asignatura, incluiremos soporte para fuentes de luz direccionales y puntuales, y los materiales contendrán propiedades de color difuso e intensidad especular. Además, se podrá definir en el mundo el valor de la luz ambiente global.

Clase Vertex

Debemos añadir a los vértices información de la normal del mismo. Esta información deberá ser cargada de los ficheros de malla si el buffer contiene un elemento `<normals>`.

Clase Light

Esta subclase de Entity contiene información sobre una fuente de luz. Necesitaremos los siguientes métodos:

```
Type                getType() const;
void                setType(Type type);
const glm::vec3&    getColor() const;
void                setColor(const glm::vec3& color);
float               getLinearAttenuation() const;
void                setLinearAttenuation(float att);
void                prepare(int index, std::shared_ptr<Shader>& shader) const;
```

El tipo de luz puede ser `DIRECTIONAL` o `POINT`, y deberíamos definirlo como un enumerado dentro de la clase. Podemos definir la atenuación lineal de la luz, aunque por simplificación los valores de atenuación constante y cuadrática no serán modificables y tendrán los valores 1 y 0 respectivamente.

El método `prepare` será invocado por cada material para todas las luces antes de realizar el renderizado. El objetivo del método será escribir en el shader recibido como parámetro las variables uniformes con la información de la luz. Los datos de la luz se pasan al shader **en espacio del observador**.

Clase State

Vamos a añadir la dos siguientes variables a la clase:

```
static std::vector<std::shared_ptr<Light>>      lights;
static glm::vec3                                ambient;
```

La primera contendrá las luces activas del mundo, y la segunda el valor de luz ambiente. Ambas variables serán escritas por el mundo antes del pintado, de forma que el resto de componentes del motor tendrán esta información accesible.

Clase Shader

Debemos añadir una nueva variable attribute al vertex shader para la normal del vértice (por ejemplo, `vnormal`). El método `Shader::setupAttribs` debe activar su escritura y configurar la disposición de los datos en el buffer.

En el fragment shader, calcularemos la iluminación de la forma siguiente (añadiremos todas las variables que sean necesarias):

Creamos dos variables **vec3** para **acumular por separado la contribución difusa y especular** de las fuentes de luz. Esto se hace así porque la difusa se multiplica por los valores del material (color, textura, etc), mientras que el especular debe sumarse al final. Si el número de luces en la escena es mayor a cero, inicializamos el difuso con la luz ambiente; el especular a (0,0,0). Para cada fuente de luz, haremos lo siguiente:

- Obtenemos el vector **`N`** pasando la normal del vértice al espacio del observador y normalizando (se debería hacer en el vertex shader y pasarlo al fragment, donde se debe hacer la normalización).
- Obtenemos el vector **`L`** con la `xyz` del vector de la luz que estamos calculando (esta información ya se pasa al shader en espacio del observador).
- Inicializamos a 1 una variable en la que guardaremos el factor de atenuación.
- Si la luz es puntual (es decir, si la `w` del vector de la luz vale 1), restamos al vector `L` la coordenada del vértice en espacio del observador, y calculamos el factor de atenuación con la fórmula vista en las diapositivas (la distancia de la superficie a la luz la podemos obtener con `length(L)`).

- Normalizamos el vector `L`, y obtenemos el producto escalar entre `N` y `L`. El valor debe ser positivo, así que utilizaremos 0 si el valor es negativo (podemos usar la función `max`).
- Acumulamos al difuso la incidencia de esta luz, que se calcula con el producto escalar de `N` y `L` multiplicado por el color de la luz y la atenuación.
- Queda por calcular la componente especular (que únicamente se calculará si la atenuación especular pasada desde el motor tiene un valor superior a 0, y si el coseno -producto escalar- de `N` y `L` es superior a 0). Calcularemos el vector **`H`** normalizando la diferencia entre `L` y la posición del vértice es espacio del observador (y como `L` es normal, normalizamos la posición también). Obtenemos el producto escalar de `N` y `H` (nuevamente debe ser positivo), y acumulamos al especular el resultado de la siguiente fórmula:

```
pow(NdotH, shininess) * attenuation
```

Una vez se han calculado todas las luces, tenemos por un lado la contribución difusa y ambiente de las luces, y por otro la especular. Si hay luces en la escena, la forma de calcular gl_FragColor cambia, ya que multiplicaremos el color del fragmento (teniendo en cuenta su difuso y textura si la utiliza) por la contribución difusa y ambiente de las luces, y le sumaremos la contribución especular. Si no hay fuentes de luz, el color del fragmento será el color difuso del material multiplicado por la textura (si se está utilizando).

> **NOTA:** Podemos definir arrays en GLSL de la misma forma que en C. Para obtener la localización, podemos obtener la de cada uno de sus elementos con "arrayName[0]", "arrayName[1]", etc.

Clase World

Esta clase necesitará métodos para obtener y establecer el valor de luz ambiente:

```
const glm::vec3&  getAmbient() const;
void              setAmbient(const glm::vec3& ambient);
```

Además, añadiremos una lista de luces. En el método `addEntity`, de la misma forma que comprobamos si la entidad es una cámara y la añadimos a una lista adicional en ese caso, ahora haremos lo propio con las luces que se añadan. Debemos hacer la operación contraria en `removeEntity`.

En el método draw, antes de iterar por cada cámara y dibujar las entidades, daremos valor a las nuevas variables de la clase `State` (para que esta información esté actualizada y disponible para todas las entidades que se pinten).

Clase Material

Los materiales tienen ahora dos nuevas propiedades, color y brillo (velocidad de atenuación de la intensidad especular). Los estableceremos con los siguientes métodos:

```
const glm::vec4&  getColor() const;
void              setColor(const glm::vec4& color);

uint8_t           getShininess() const;
void              setShininess(uint8_t shininess);
```

El elemento del material en los ficheros XML de malla tendrán además dos subelementos, `<color>` y `<shininess>`. El primero es un vector de cuatro coordenadas (valores separados por coma; el cuarto valor no se utiliza por ahora, pero define la opacidad del objeto). El segundo será un valor numérico entero entre 0 y 255.

El método `prepare` necesitará algunos cambios. Necesitaremos escribir en el shader dos nuevas matrices:

- Una matriz **ModelView** que lleve el vértice al espacio del observador, donde calcularemos la iluminación.
- Una matriz **Normal**, que se utiliza para corregir algunas deformaciones que pueden sufrir las normales con escalados no uniformes. Esta matriz es la **traspuesta de la inversa** de la matriz que lleva al espacio donde se realiza la iluminación (en este caso, la matriz ModelView). Los vectores normales no se pasan al espacio del observador entonces utilizando la matriz ModelView, sino utilizando la matriz normal. Podemos encontrar más información aquí.

También tenemos que pasar al shader los otros datos necesarios para el cálculo de la iluminación:

- Número de luces en el mundo
- Color difuso del material activo
- Intensidad especular (shininess) del material activo
- Valor de luz ambiente del mundo
- Información de cada luz (llamaremos al método `prepare` de la luz para hacer esto, pasándole además el índice de la luz para escribir los datos en el índice correcto del array en el shader).

Algunos de estos datos están en el propio material, otros están en la clase `State`.

Programa principal

Mostraremos en la escena la malla **“data/bunny.msh.xml”**. Definiremos como luz ambiente de la escena el color (0.2, 0.2, 0.2). Crearemos una fuente de luz direccional, de color blanco, con la dirección 1,1,1, y una luz puntual de color rojo. La luz puntual debe orbitar alrededor del modelo, a una distancia de 5 puntos, y tendrá un valor de atenuación lineal de 0.2.