Iluminación II

A. Gavilanes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

La clase Light

```
class Light { // Abstract class
   protected:
      static GLuint cont; // Atributo para poder generar un nuevo id cada vez
      GLuint id = GL_LIGHT0 + GL_MAX_LIGHTS; // Primer id no válido
      // Atributos lumínicos y geométrico de la fuente de luz
      glm::fvec4 ambient = {0.1, 0.1, 0.1, 1};
      glm::fvec4 diffuse = {0.5, 0.5, 0.5, 1};
      glm::fvec4 specular = {0.5, 0.5, 0.5, 1};
      glm::fvec4 posDir = {0, 0, 1, 0};
      // Añade setter's para cambiar el valor de los atributos lumínicos
   public:
      Light();
      virtual ~Light() {disable();}
      void uploadL();
      // Método abstracto
      virtual void upload(glm::dmat4 const& modelViewMat) const = 0;
 … };
```

La clase Light

- Funcionalidad de la clase

```
GLuint Light::cont = 0;
Light() {
    if (cont < GL_MAX_LIGHTS) {
            id = GL_LIGHT0 + cont;
            ++cont;
            glEnable(id);}};
void Light::uploadL() {
     // Transfiere las características de la luz a la GPU
     glLightfv(id, GL_AMBIENT, value_ptr(ambient));
     glLightfv(id, GL_DIFFUSE, value_ptr(diffuse));
     glLightfv(id, GL_SPECULAR, value_ptr(specular));
}
void disable() {if (id<GL_LIGHT0+GL_MAX_LIGHTS) glDisable(id);};
void enable() {if (id<GL_LIGHT0+GL_MAX_LIGHTS) glEnable(id);};
void setAmb(glm::fvec4 amb){ambient = amb; upLoadL();}; //setDiff(), setSpec()
```

La clase DirLight

```
class DirLight : public Light {
    public:
        virtual void upload(glm::dmat4 const& modelViewMat) const;
        void setPosDir(glm::fvec3 dir);
};
```

donde:

```
void DirLight::upload(glm::dmat4 const& modelViewMat) const {
        glMatrixMode(GL_MODELVIEW);
        glLoadMatrixd(value_ptr(modelViewMat));
        glLightfv(id, GL_POSITION, value_ptr(posDir));
        uploadL();
}
// Ojo al 0.0 que determina que la luz sea remota
void DirLight::setPosDir(glm::fvec3 dir); {
        posDir = glm::fvec4(dir, 0.0);
}
```

La clase PosLight

```
class PosLight : public Light {
  protected:
    // Factores de atenuación
    GLfloat kc = 1, kl = 0, kq = 0;
  public:
    virtual void upload(glm::dmat4 const& modelViewMat) const;
    void setPosDir(glm::fvec3 dir);
    void setAtte(GLfloat kc, GLfloat kl, GLfloat kc);
};
```

- donde:

```
void PosLight::upload(glm::dmat4 const& modelViewMat) const {
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixd(value_ptr(modelViewMat));
    glLightfv(id, GL_POSITION, value_ptr(posDir));
    glLightf(id, GL_CONSTANT_ATTENUATION, kc);
    glLightf(id, GL_LINEAR_ATTENUATION, kl);
    glLightf(id, GL_QUADRATIC_ATTENUATION, kq);
    uploadL();
}
// Ojo al 1.0 que determina que la luz sea local
void PosLight::setPosDir(glm::fvec3 dir); {
    posDir = glm::fvec4(dir, 1.0);
}
```

La clase SpotLight

```
class SpotLight : public PosLight {
    protected:
        // Atributos del foco
        glm::fvec4 direction = { 0, 0, -1, 0 };
        GLfloat cutoff = 180;
        GLfloat exp = 0;
    public:
        SpotLight(glm::fvec3 pos = { 0, 0, 0 })
              : PosLight() {posDir = glm::fvec4(pos, 1.0);};
        virtual void upload(glm::dmat4 const& modelViewMat) const;
        void setSpot(glm::fvec3 dir, GLfloat cf, GLfloat e);
};
```

- donde:

```
void SpotLight::upload(glm::dmat4 const& modelViewMat) const {
    PosLight::upload(modelViewMat);
    glLightfv(id, GL_SPOT_DIRECTION, value_ptr(direction));
    glLightf(id, GL_SPOT_CUTOFF, cutoff);
    glLightf(id, GL_SPOT_EXPONENT, exp);
 }
// Ojo al 0.0: la dirección de emisión del foco es vector
void setSpot(glm::fvec3 dir, GLfloat cf, GLfloat e) {
    direction = glm::fvec4(dir, 0.0);
    cutoff = cf;
    exp = e;
}
```

La clase Material

```
class Material {
    protected:
        // Coeficientes de reflexión
        glm::fvec4 ambient = {0.2, 0.2, 0.2, 1.0};
        glm::fvec4 diffuse = {0.8, 0.8, 0.8, 1.0};
        glm::fvec4 specular = {0.0, 0.0, 0.0, 1.0};
        GLfloat expF = 0;  // Exponente para la reflexión especular
        GLuint face = GL_FRONT_AND_BACK;
        GLuint sh = GL_SMOOTH;  // Tipo de matizado
    public:
        Material() {};
        virtual ~Material() {};
        virtual void upload();
        void setCopper();
};
```

- donde:

```
void Material::upload() {
  glMaterialfv(face, GL_AMBIENT, value_ptr(ambient));
  glMaterialfv(face, GL_DIFFUSE, value_ptr(diffuse));
  glMaterialfv(face, GL_SPECULAR, value_ptr(specular));
  glMaterialf(face, GL_SHININESS, expF);
  glShadeModel(sh);
  //glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE);  // Defecto
}
void Material::setCopper() {
  ambient = {0.19125, 0.0735, 0.0225, 1};
  diffuse = {0.7038, 0.27048, 0.0828, 1};
  specular = {0.256777, 0.137622, 0.086014, 1};
  expF = 12.8;
}
```

La clase EntityWithMaterial

```
class EntityWithMaterial : public Abs_Entity {
    public:
        EntityWithMaterial () : Abs_Entity() { };
        virtual ~EntityWithMaterial () { };

        void setMaterial(Material* matl) {material = matl;};
    protected:
        Material* material = nullptr;
};
```

Clases modificadas para manejar la iluminación

- La clase **`Scene`** contiene gran parte de las luces de la escena en forma de atributos
- Las luces de la escena se crean, y se especifican sus características, en **`setLights()`** de **`Scene`**
- En **`render()`** de **`Scene`** se hace **`upload()`** de todas ellas
- El resto de las luces aparecen en la escena porque suelen formar parte de entidades; por ejemplo, la luz de una lámpara de una habitación, los focos de un coche, o la luz de exploración de una nave sobre un planeta
- Estas luces se declaran como atributos de la entidad de la que forman parte y se construyen y establecen sus características en la constructora de esa entidad
- Si una luz está asociada a una entidad y se mueve de forma solidaria con la entidad (por ejemplo, los focos de los faros de un coche, el foco de exploración de una nave, etc.), debe fijarse su posición en el **`render()`** de la entidad (ver luces dinámicas, más adelante)

Observaciones

- Los cálculos de la iluminación en OpenGL se realizan en coordenadas de cámara. Como se verá después, según las luces sean estáticas o dinámicas, se debe tener en cuenta por tanto si la matriz de modelado-vista ha cambiado o no.
- Hay que establecer las fuentes de luz y encenderlas antes de renderizar los objetos que van a iluminar.
- Un objeto se puede dejar oscuro si se renderiza antes de que actúen las fuentes de luz que lo pueden iluminar.
- Los focos deben iluminar vértices de una malla para que se vea el cono de emisión. Si no lo hacen su luz no aparece.
- Para no alterar los colores de la textura se suelen utilizar materiales grises.
- Los colores de la textura se suelen mezclar con el de la iluminación con el comando:

```
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, mix);
```

con **mix**: **GL_MODULATE** (o **GL_REPLACE**, **GL_ADD**, **GL_SUBTRACT**, **GL_DECAL**, …)

Tipos de luces en una escena

- **Estáticas (o fijas)**
  - Son las luces, ya sean remotas, locales, y dentro de estas, focos o no, que no cambian su posición durante la renderización de la escena
  - La forma de comprobar que una luz es estática consiste en mover la cámara y ver que los efectos de la luz no cambian
  - La posición debe restablecerse cada vez que cambie la matriz de modelado-vista
  - Si la cámara no se va a mover, su posición se fija solo una vez en **`setLights()`** de **`Scene`**
  - Si la cámara se va a mover, hay que volver a fijar la posición de la luz cada vez que se mueva. Por ello, al principio de **`render(cam)`** de **`Scene`**, la luz tiene que llamar a **`upload(cam.viewMat())`** donde lo primero que se hace es cargar la matriz de modelado-vista y después se fija la posición de la luz

Tipos de luces

- **Dinámicas**
  - Luces que cambian de posición, independientemente de la cámara
    - **Focos en objetos de la escena que se mueven** (por ejemplo, la luz de exploración debajo de un avión)
    - La posición inicial se fija en la constructora de la entidad que las contiene; generalmente esa posición suele ser el punto (0, 0, 0)
    - La posición se actualiza después de multiplicar por la matriz **`modelMat`**, en el **`render()`** de la entidad que las contiene
  - Luces que se mueven con la cámara
    - **Luz de minero posicional (que puede ser foco o no) en la cámara**
    - La posición se establece una vez, después de que la matriz de modelado-vista se ha fijado
    - Si se trata de un foco colocado en la cámara, su dirección de emisión es (0, 0, -1)
    - La posición se fija en **`setLights()`** de **`Scene`** y se actualiza en **`render()`** de **`Scene`** llamando con ella a **`upload(dmat4(1.0))`**