Taller #3 - Despliegue de Modelos 3D

Jesús Cibeira

Noviembre 2018

Agenda

- Bibliotecas
- Pipeline gráfico
- Formatos de archivos de objetos 3D
- Modos de Despliegue
- Transformaciones afines
- Normales
- Estilo de despliegue
- Matrices
- Explicación de código

Bibliotecas

- **GLEW** (Biblioteca de Extensión de OpenGL) es multiplataforma y de código abierto escrita en C/C++, proporciona mecanismos de ejecución eficientes para determinar qué extensiones de OpenGL se admiten en la plataforma de destino.
- **GLM** (OpenGL Matemáticas) únicamente para C++, provee diversas estructuras matemáticas y sus respectivas operaciones basadas en las de GLSL.

Pipeline gráfico

La denominación pipeline hace referencia a que las etapas se encuentran secuenciadas y que el conjunto de ellas puede verse como una línea de producción de una fábrica. El pipeline gráfico puede modelarse de manera genérica en tres etapas descritas a continuación:

Transformaciones

- Mundo
- Vista
- Proyección

Ensamblado, recorte y mapeo a pantalla

Rasterización

Formatos de archivos de objetos 3D

- OBJ
  - **v**: representa los vértices del modelo.
  - **vn**: representa las normales del modelo.
  - **vt**: representa las coordenadas de textura del modelo (formato u v).
  - **f** : representa las caras del modelo.
- OFF
  - La primera línea representa el **número de vértices**, **número de caras** y **número de aristas**.
  - Seguido se encuentra un lista de vértices y de caras (**índices**, en ocasiones con otros datos como el **color**).

Modos de Despliegue

- glBegin() glEnd()

```
void model::display() {
	for (int i = 0; i < numOfIndex; i += 3) {
		glVertex3f(vertices[index[i].x].x, vertices[index[i].y].y, vertices[index[i].z].z);
		glVertex3f(vertices[index[i + 1].x].x, vertices[index[i + 1].y].y, vertices[index[i + 1].z].z);
		glVertex3f(vertices[index[i + 2].x].x, vertices[index[i + 2].y].y, vertices[index[i + 2].z].z);
	}
}
```

Modos de Despliegue

- Display List

```
void model::createList() {
        GLuint i;
        list = glGenLists(1);
        glNewList(list, GL_COMPILE);
        glColor3fv(color);
                glPolygonMode(GL_FRONT_AND_BACK, GL_POINTS);
                glBegin(GL_POINTS);
                        display();
                glEnd();
        glEndList();
}

void model::displayList() {
        glCallList(list);
}
```

Modos de Despliegue

- Vertex Buffer Object
  - Creación

```
void model::createVBO() {
        glGenBuffers(1, &vbo);
        glGenBuffers(1, &vindex);

        glBindBuffer(GL_ARRAY_BUFFER, vbo);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vindex);
        glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat)* numOfVertex * 3, vertex, GL_STATIC_DRAW);
        glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(GLuint)* numOfIndex, index, GL_STATIC_DRAW);
        glBindBuffer(GL_ARRAY_BUFFER, 0);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
}
```

Modos de Despliegue

- Vertex Buffer Object
  - Despliegue

```
void model::displayVBO() {
        glBindBuffer(GL_ARRAY_BUFFER, vbo);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vindex);
        glDrawElements(GL_TRIANGLES, mNumOfIndex, GL_UNSIGNED_INT, 0);
        glBindBuffer(GL_ARRAY_BUFFER, 0);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,                                    0);
}
```

Transformaciones afines

- Traslación

$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde T_x, T_y y T_z representan la traslación aplicada al modelo.

Transformaciones afines

- Rotación
  - Realizada mediante cuaterniones.
  - Transformar el cuaternión a matriz para realizar la rotación.

$$R_X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_z = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde α representa el ángulo de rotación del modelo.

Transformaciones afines

- Rotación

Un cuaternión está representado por $q_w + i\ q_x + j\ q_y + k\ q_z$, entonces para acumular la rotación, se utiliza lo siguiente:

this->rotation = **glm::quat**(1.0f, 0.0f, 0.0f, 0.0f);

quaternion = **glm::angleAxis**(angle, axis);

this->rotation = **glm::cross**(quaternion, this->rotation);

Transformaciones afines

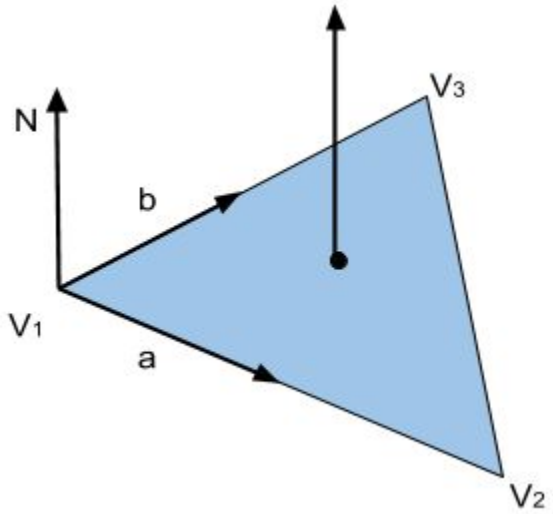
- Escalamiento

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde S_x, S_y y S_z representan el escalamiento aplicado al modelo.

Normales

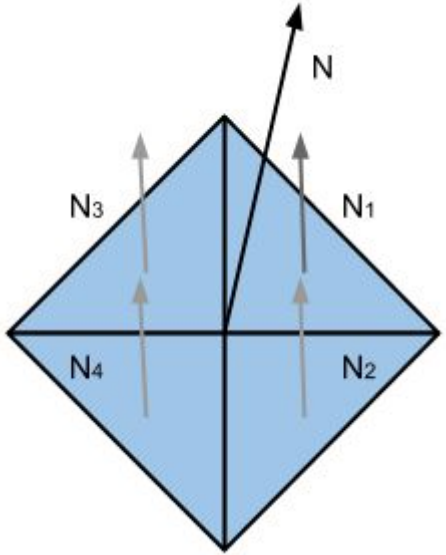
- Por cara



Donde V_1, V_2 y V_3 representan los vértices de un triángulo cualquiera

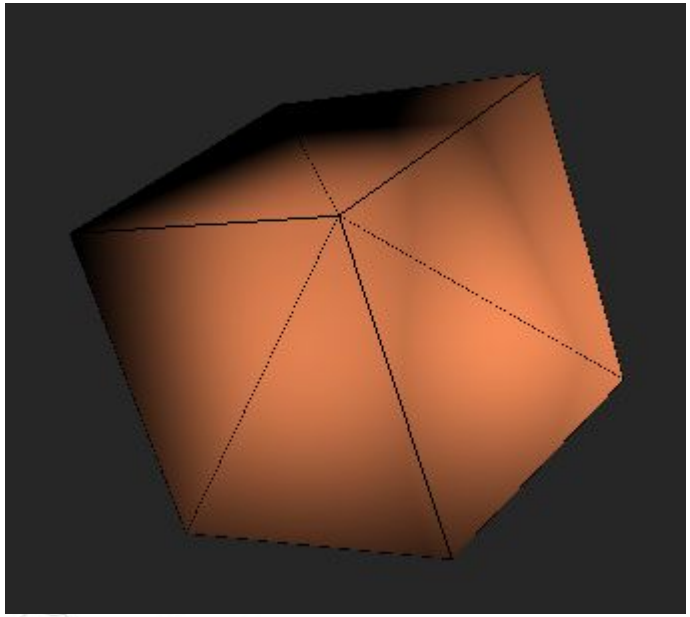
Normales

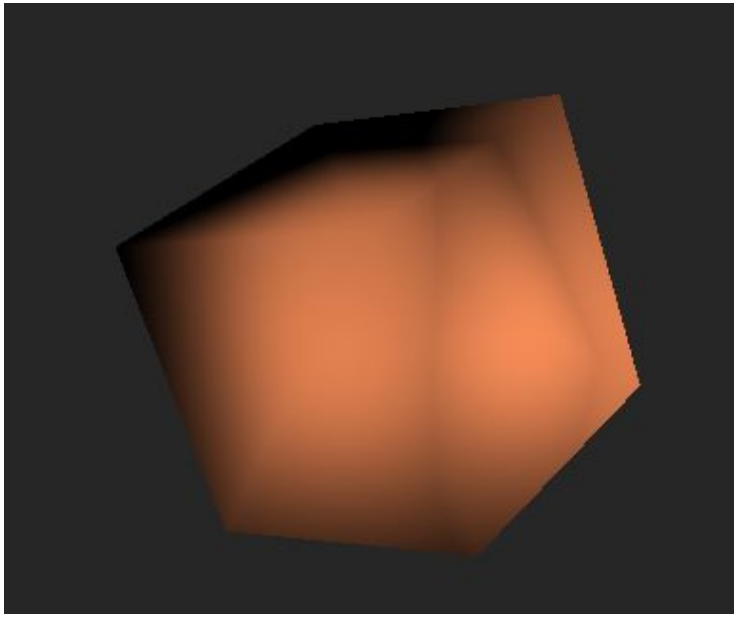
- Por Vértice



Donde N_1, N_2, N_3 y N_4 representan normales de triángulos cualesquiera

Estilo de despliegue





Matrices

- Matriz de Modelo

Es aquella matriz que contiene las transformaciones afines que aplicaron al modelo. Es calculada de la siguiente manera:

MModelo = MTraslación * MRotación * MEscalamiento

Matrices

- Matriz de Vista

La matriz de vista define la manera en que vemos una escena. Para el proyecto debemos usar lo siguiente:

```
glm::lookAt(glm::vec3(0.0f, 0.0f, 5.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
```

Donde el primer parámetro representa la **posición del ojo**, el segundo el **centro de la escena** y el último representa el **up**, es decir cuál cuadrante es la parte de arriba de la escena.

Matrices

- Matriz de Proyección
  - Ortogonal:

```
glOrtho(0.0f, (float)gWidth / 100.0f, 0.0f, (float)gHeight / 100.0f, 0.1f, 100.0f);
```

  - Perspectiva:

```
gluPerspective(45.0f, (float)gWidth / (float)gHeight, 0.1f, 100.0f);
```

Explicación de código