UNIVERSIDAD
NACIONAL
DE COLOMBIA

Taller Final: Interacción y Shaders

Jose Iván Carpintero

Computación Visual
Bogotá, 2019

Introducción

En el taller de Interacción se implementó una aplicación de Cámara en Tercera Persona, el cual fue desarrollado en el entorno de Processing, utilizando el modo P5.js, empleando el lenguajes JavaScript, para la implementación se utilizo la librería three.js.

En el taller de Shaders se implementó el shadow mapping, el cual es el mapeo de sombras o la proyección de sombras es un proceso mediante el cual se agregan sombras a los gráficos de computadora en 3D. El cual fue desarrollado en el entorno de Processing, utilizando el modo P5.js, empleando el lenguajes JavaScript.

Objetivos

- Estudiar e Implementar Interacciones en entornos virtuales y mostrar vistas de cámara en tercera persona, de algún objeto.

- Estudiar e Implementar patrones de diseño de shaders, enfocados a la generación de sombras.

Desarrollo - Cámara en Tercera Persona

- Calcular las matriz mundo de cada objeto. Es aquí donde el grafo de escena entra a jugar, calculando las matrices de mundo de los objetos según sus objetos padre.
- Calcular la matriz de cámara, según la posición y dirección de la escena donde queremos que nuestra cámara este.
- Calcular la matriz vista. La cual es la matriz inversa a la matriz de cámara, de esta forma estamos moviendo todo el mundo alrededor de la cámara y no la cámara alrededor del mundo.

Desarrollo - Cámara en Tercera Persona

- Calcular la matriz de proyección. Con esta definimos si el objeto es una cámara de perspectiva, además podemos ajustar el ángulo de visión de la misma y los planos cerca y lejos.
- Calcular la matriz de Vista-Proyección. La cual es la multiplicación de la matriz de proyección y vista.
- Multiplicar las matriz mundo de cada objeto por la matriz de proyección vista. Dibujar los objetos.

Desarrollo - Cámara en Tercera Persona

```
var view = document.getElementById('view').innerHTML;
  var cameraMatrix = m4.identity();
  // Cambiamos la camara segun la vista
  if (view == "first") {
    m4.copy(targetNode.worldMatrix, cameraMatrix);
    cameraMatrix = m4.translate(cameraMatrix, 0, 0, 0);
    cameraMatrix = m4.xRotate(cameraMatrix, degToRad(90));
    cameraMatrix = m4.yRotate(cameraMatrix, -degToRad(90));
  } else if (view == "third") {
    m4.copy(targetNode.worldMatrix, cameraMatrix);
    cameraMatrix = m4.translate(cameraMatrix, -50, 0, 10);
    cameraMatrix = m4.xRotate(cameraMatrix, degToRad(90));
    cameraMatrix = m4.yRotate(cameraMatrix, -degToRad(90));
  } else if (view == "long") {
    var cameraPosition = [0, -200, 0];
    var target = [0, 0, 0];
    var up = [0, 0, 1];
    cameraMatrix = m4.lookAt(cameraPosition, target, up);
  }
```

Desarrollo - Shadow Mapping

Se crea un mapa de sombras de la escena por cada frame.

```
uniform mat4 mProj;
uniform mat4 mView;
uniform mat4 mWorld;

attribute vec3 a_position;
attribute vec3 a_normal;

varying vec3 fPos;
varying vec3 fNorm;

void main()
{
  fPos = (mWorld * vec4(a_position, 1.0)).xyz;
  fNorm = (mWorld * vec4(a_normal, 0.0)).xyz;

  gl_Position = mProj * mView * vec4(fPos, 1.0);
}
</script>
```

Desarrollo - Shadow Mapping

En el vertex shader, se obtiene la posición del vértice y las matrices de proyección, vista y mundo.

```
uniform vec3 pointLightPosition;

varying vec3 fPos;

void main()
{
    vec3 fromLightToFrag = (fPos - pointLightPosition);

    float lightFragDist =length(fromLightToFrag);

    gl_FragColor = vec4(lightFragDist, lightFragDist, lightFragDist, 1.0);
}
```

Desarrollo - Shadow Mapping

En el vertex shader, obtenemos la posición del vértice y las matrices de proyección, vista y mundo.

```
uniform vec3 pointLightPosition;

varying vec3 fPos;

void main()
{
    vec3 fromLightToFrag = (fPos - pointLightPosition);

    float lightFragDist =length(fromLightToFrag);

    gl_FragColor = vec4(lightFragDist, lightFragDist, lightFragDist, 1.0);
}
```

Desarrollo - Shadow Mapping

Al fragment shader le pasamos la posición calculada en el vertex shader y la posición de la luz en el mundo.

```
uniform mat4 mProj;
uniform mat4 mView;
uniform mat4 mWorld;

attribute vec3 a_position;
attribute vec3 a_normal;

varying vec3 fPos;
varying vec3 fNorm;

void main()
{
    fPos = (mWorld * vec4(a_position, 1.0)).xyz;
    fNorm = (mWorld * vec4(a_normal, 0.0)).xyz;

    gl_Position = mProj * mView * vec4(fPos, 1.0);
}
```

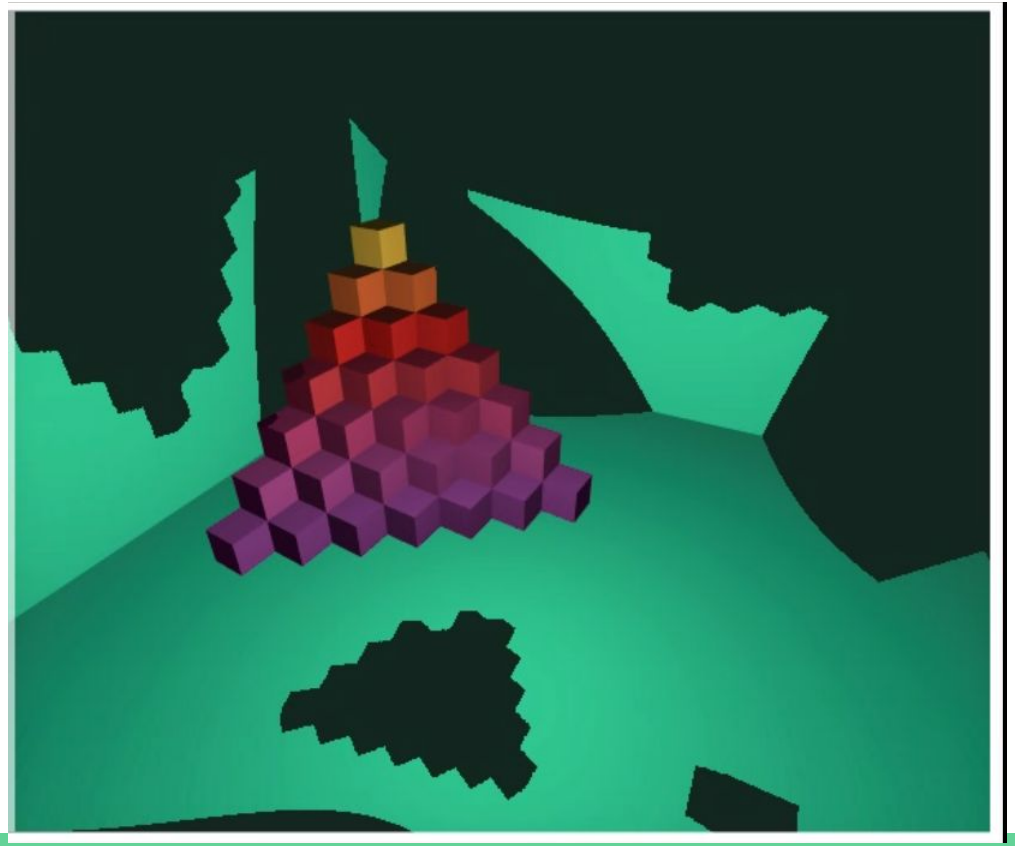
Desarrollo - Shadow Mapping

En el fragment shader realizaremos el cálculo de las sombras. La textura creada anteriormente al fragment shader por medio de un sampler.

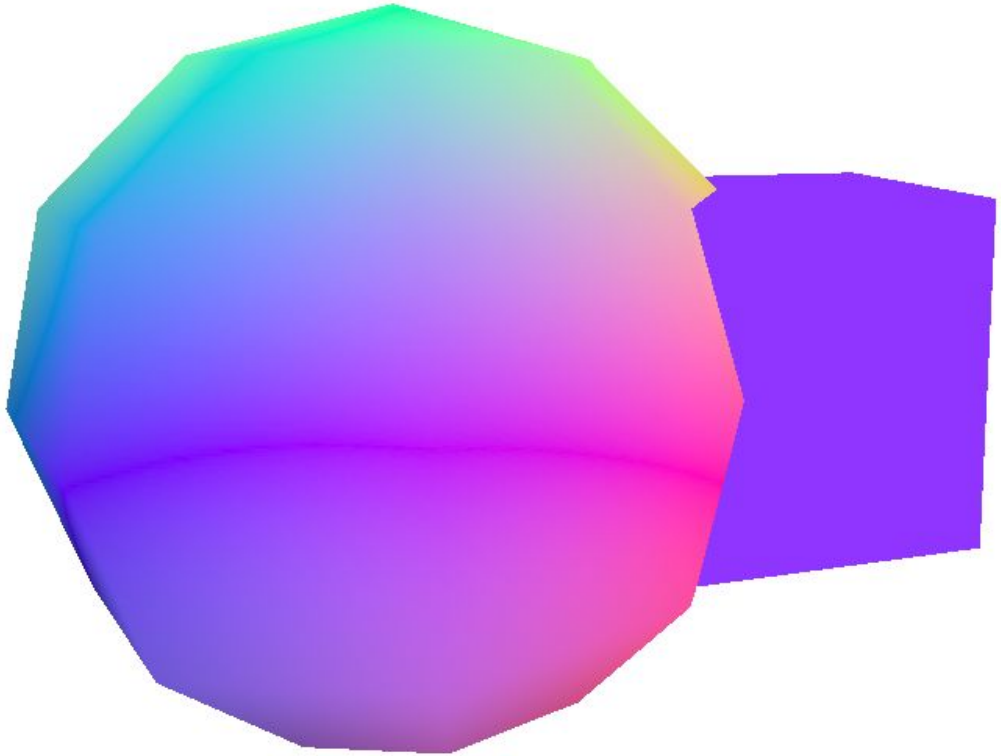
Con el shaders modificado, ahora se tiene que implementar el código para crear la textura del mapa de sombras.

Ya que se va a crear el mapa de sombras para las seis direcciones del espacio. se debe realizar 6 pasos a la escena utilizando cámaras que estén en la posición de la luz y apuntando en las direcciones especificas (x positivo, x negativo, y positivo, etc.).

Demo - Shadow Mapping



Demo - Interacción



Conclusiones

- Los mapas de sombras son un valioso enfoque para generar sombras complejas en condiciones razonables.
- El primer enfoque es Simple Shadow Mapping, incorpora el renderizado a una textura de profundidad fuera de la pantalla desde el punto de vista de la luz.
- La cámara en tercera persona es una de las vistas más frecuentes de los videojuegos actuales. Esta vista tiene como característica que el personaje que se controla se ve de cuerpo entero, el cual hace que interacción sea más entretenida.

Referencias

https://en.wikipedia.org/wiki/Shadow_mapping

https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping

https://medium.com/cegonzalez13/shadows-in-webgl-e7498cdfc46c

https://webglfundamentals.org/webgl/lessons/webgl-3d-camera.html

https://github.com/mrdoob/three.js/tree/master