Lenguaje de shaders

Cristian Camilo Galeano [1]

Daniel Leonardo Bustos [2]

Andres Felipe Alvarado [3]

25 de Septiembre del 2020

Computación gráfica

u6000413@unimilitar.edu.co, u6000411@unimilitar.edu.co , u6000399@unimilitar.edu.co

Resumen: Three.js al ser un librería que permite crear escenarios 3D, también permite el uso de shader, estos shaders estarán basados en una leguaje de OpenGL, el cual darán a los objetos un toque realista dentro de una escena. Este trabajo escrito definirán los conceptos esenciales que serán usados para aplicar el lenguaje de shaders en three.js.

Palabras clave: Shader, Vertex Shader, Pixel Shader, uniformes, atributos y variaciones.

I. Introducción

El siguiente trabajo tiene como propósito definir el lenguaje de shaders que es usado en three.js, pero primero que todo hay que entender que son los shaders, los tipos de shaders que existen y su importancia en la composición de escenarios 3D, además del uso de nuevas variables exclusivas para shaders. Se comprenderá de manera práctica el uso de los tipos de shaders y su relevancia en la computación gráfica, la simulación por computadora y desarrollo de videojuegos.

Se observará que hay una relación entre WebGLrenderer y el uso de shaders en three.js, WebGLrenderer mostrará las escenas elaboradas en la Api de javaScript WebGL, el cual se ejecutará en la GPU (graphics processing unit) del ordenadorz. WebGL estará a su vez relacionado con el programa que sombreara las escenas en three.js estará escrito en GLSL o OpenGL Shading Language , este lenguaje nos dará un acceso mucha más directa al pipeline de gráficos, sin necesidad de acudir a otro tipo de lenguajes mucho más complejos.

II. Marco teórico

¿Qué son los shaders?

La tecnología shaders es cualquier unidad, escrita en un lenguaje de sombreado que se puede compilar independientemente.
Es una tecnología reciente y que ha experimentado una gran evolución destinada a proporcionar al programador una interacción con la unidad de procesamiento gráfico (GPU) hasta ahora imposible.
Los shaders son utilizados para realizar transformaciones y crear efectos especiales, como por ejemplo iluminación, fuego o niebla.

¿Qué es una GPU?

GPU es el acrónimo de Graphics Processing Unit y representa precisamente el corazón de una tarjeta gráfica al igual que la CPU lo hace en un PC. Aparte del corazón, también es su cerebro, ya que es la encargada de realizar todos los cálculos complejos que nos permiten disfrutar de nuestros juegos en pantalla. En definitiva es la pieza de silicio que tanto AMD

como NVIDIA o Intel fabrican y donde se graban los transistores.

¿Qué tipos de shaders hay?

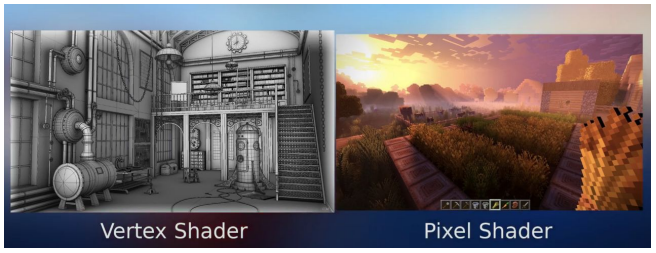
- Vertex shader

Es una instrucción encargada de calcular los parámetros de los vértices de cada uno de los millones de triángulos que forman un objeto 3D de un juego.

Con esta herramienta, los programadores de juegos cuentan con la posibilidad de personalizar y reprogramar la información que recoge cada uno de los tres vértices que componen los triángulos o polígonos, que a su vez forman las figuras 3D.

- Pixel shader

Son programas que se ejecutan mediante el procesador gráfico de una tarjeta gráfica 3d en el transcurso gráficos (en las llamadas unidades de sombreado).



Hay tres tipos de variables en los sombreadores: uniformes, atributos y variaciones:

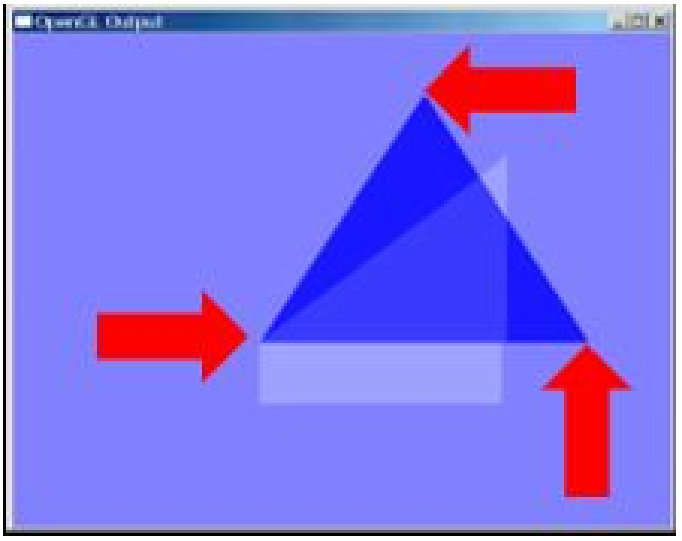
- Los uniformes

Son variables que tienen el mismo valor para todos los vértices; los mapas de iluminación, niebla y sombras son ejemplos de datos que se almacenarán en uniformes. Se puede acceder a los uniformes mediante el sombreador de vértices y el sombreador de fragmentos.

1.Imagen, Uniform Variables.

- Los atributos

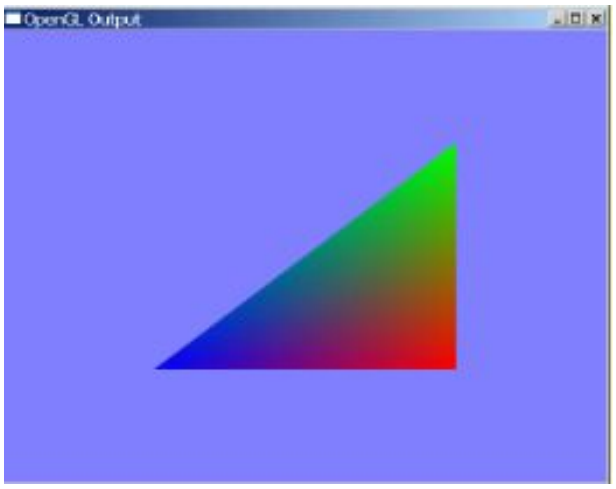
Son variables asociadas con cada vértice; por ejemplo, la posición del vértice, la cara normal y el color del vértice son ejemplos de datos que se almacenarán en atributos. Solo se puede acceder a los atributos dentro del sombreador de vértices.



2.Imagen, Uniform Atributos.

- Las variaciones

Son variables que se pasan del sombreador de vértices al sombreador de fragmentos. Para cada fragmento, el valor de cada variable se interpolara suavemente a partir de los valores de los vértices adyacentes.



2.Imagen, Uniform Variaciones.

III. Desarrollo

Para realizar este ejercicio se necesita de las siguientes librerías :

- Three.js: Esta nos servirá para poder aplicarle los shaders a lo que sería nuestra escena además de permitirnos crear nuestro entorno 3D.
- OrbitControls.js: Esta nos va a permitir manejar la cámara por eventos del teclado o por la acciones del mouse.
- OBJLoader.js: Por último el object loader nos va a servir para cargar archivos de modelado 3D personalizados y agregarlos a nuestra escena.

Para que el programa funcione se debe tener estas tres librerías en nuestra carpeta de proyecto y llamarlas en el código de la siguiente forma:

```
<head>
  <title>three.js empty template</title>
  <script src="js/three.min.js"></script>
  <script src="js/OBJLoader.js"></script>
  <script src="js/controls/OrbitControls.js"></script>
  <style>
```

Luego de tener las librerías y nuestra escena creada el paso a seguir es cargar el objeto 3D a la escena de la siguiente forma :

```
///load monkey object
   var loader = new THREE.OBJLoader();
   var monkeyMesh;
   loader.load( 'assets/monkey.obj', function ( object ) {
     object.traverse( function ( child ) {
       if ( child instanceof THREE.Mesh ) {
         console.log(child.geometry)
         var geo1 = child.geometry;
         monkeyMesh = new THREE.Mesh(geo1, monkeyMaterial);
       }
     } );
     scene.add( monkeyMesh );
   });
```

Lo que se hace aquí es crear una variable loader que permita cargar el archivo .obj y guardarlo en una variable objeto. Para esto se usa en el constructor **load()**, en donde primero se le indica en donde se encuentra el objeto 3D y luego se encapsula ese objeto en una variable, luego de esto se le indica al objeto que atraviese toda la geometría seleccionando todas las mallas poligonales en ellas y las guarde en una variable tipo **mesh(monkey mesh)**. Posteriormente se agregara la geometría a la escena quedando el siguiente resultado.

Ahora se quiere lograr hacer uso de shaders para ponerle sombras y modificar su geometría, para esto tenemos que crear un nuevo material que hará el papel de textura para el modelo objeto 3D.

Se crea la variable **monkeymaterial** del tipo **shader material** y es donde se modificara los sombreadores del objeto, tales como los **uniforms, el vertex shader, y el fragment shader** pero en este ejemplo no se hará tanto énfasis en el **fragment shader**.

Para ponerle sombras a nuestro mono se hizo uso de **phongshader** que es una clase de **three js** que ya tiene

```
//define monkey monkeyMaterial
var monkeyMaterial = new THREE.ShaderMaterial ({
 uniforms : mUniforms,
  vertexShader: document.getElementById( 'vertexShader' ).textContent,
fragmentShader: phongShader.fragmentShader,
  lights: true,
  side : THREE.DoubleSide,
});
```

por defecto configurado el fragment shader, de tal forma que al utilizar esta función podemos generar sombras al objeto.

Posteriormente en el **uniforms** solo se definirá las variables **GLSL (OpenGL Shading Language)** del objeto, para este caso solo se modificará el color del objeto y el color de la luz que reflejará el sólido.

```
var uniforms = ({
  timeDelta : {type: 'f', value:0},
  emissive : {type: 'c', value: new THREE.Color(0xe09278)},
  specular : {type: 'c', value: new THREE.Color(0x38FD19)}
});
```

Como ya se había mencionado con **pongshader** obtenemos una configuración ya determinada para generar sombras al sombreador de píxeles del objeto.

```
53        var phongShader = THREE.ShaderLib.phong;
54
55        var mUniforms = THREE.UniformsUtils.merge([phongShader.uniforms, uniforms]);
56
```

Luego de haber modificado los uniformes y el sombreador de píxeles, se procederá a modificar el sombreador de vértices, moviendo la figura en el eje Y, produciendo además una deformación en el rostro del mono. Para modificar el sombreador de vértices se hace en un script tipo html aparte; en donde se modificara los uniforms variables GLSL para modelar los vértices de la figura.

```
</head>

<body>
  <script id="vertexShader" type="x-shader/x-vertex">

    varying vec3 vNormal;
    varying vec3 vViewPosition;
    uniform float timeDelta;

    void main() {
       vec3 centro = vec3(0.0,0.0,0.0 );
      vec3 p = position;
      float distance = length(centro - p);
      p.y += sin(distance+ timeDelta/1000.0)*0.5;
      vec4 modelViewPosition = modelViewMatrix * vec4(p, 0.5);
      vViewPosition = -modelViewPosition.xyz;
      gl_Position = projectionMatrix * modelViewPosition;
      vNormal = normalMatrix * normal;
    }
```

Aquí lo que se hizo fue modificar la posición del objeto; cada vez que cambie el tiempo la componente de la posición del objeto va a modificarse y para hacer que se deforme lo que se hace es mover el centro de la figura y aplicar ese cambio a la posición del objeto. Y por último se define la posición en donde va estar ubicada la cámara que está viendo al objeto. Una vez modificado el sombreador de vértices se ha terminado de crear el material sombreado; ahora solo lo aplicamos a la geometría.

```
monkeyMesh = new THREE.Mesh(geo1, monkeyMaterial);
```

Al finalizar se contempla el siguiente resultado :

Como se observa el objeto se mueve y se deforma en el eje y.

IV. Conclusiones

Los shaders son comúnmente conocidos en el mundo de la computación gráfica como algoritmos que calculan el color de los píxeles a partir del cambio de luz y posición de cierto objeto, además se conocen tres tipos de sombreadores que se pueden implementar y hay que tener claro que cada uno cumple con una función diferente. Además que existen diferentes variables para los sombreadores, poniendo incluso en práctica este tipo de shaders y de variables.

Este trabajo además recopiló de manera más técnica el procesamiento de sombras en la tarjeta gráfica, siendo la forma más eficiente hoy en día de procesar este tipo de sombras y para ello se expuso la importancia del WebGLrenderer y del lenguaje GLSL en el uso de shaders.

Se contempló a lo largo de toda la investigación three.js es una herramienta que ofrece una gran variedad de funciones, no solo para crear espacios tridimensionales si que también permite personalizar objetos usando shaders, y con estos se puede

modificar tanto los pixeles del objeto como también modelar los vértices de este.

VI. Bibliografía

[1] López, J. (2020, 8 julio). ¿Qué es exactamente una GPU? HZ. https://hardzone.es/reportajes/que-es/gpu-caracteristicas-especificaciones/

[2] Shaders | Minecraftpedia | Fandom. (s. f.). Minecraftpedia. Recuperado 19 de septiembre de 2020, de https://minecraft.fandom.com/es/wiki/Shaders

[3] Vertex shader - Sección Multimedia. (s. f.). Recuperado 19 de septiembre de 2020, de https://www.glosarioit.com/Vertex_shaders

[4] Flames, M. F. (2019, 21 enero). Creating a custom shader in Three.js. DEV. https://dev.to/maniflames/creating-a-custom-shader-in-threejs-3bhi

[5] OpenGL Shading Language. (2020, 5 septiembre). Wikipedia.org, de https://en.wikipedia.org/wiki/OpenGL_Shading_Language

[6] Tutorials. (s. f.). OpenGL. Recuperado 19 de septiembre de 2020, de https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/uniform.php

[7] Sneha Belkhale. (2017, 14 noviembre). three.js custom vertex shader + phong fragment shader. YouTube. https://www.youtube.com/watch?v=978-x5IL96Y&start=1642s