

Universidad La Salle Noroeste

Ingeniería en Producción Multimedia

Tópicos de Física

**Proyecto primer parcial**

Jesús Eduardo Gastélum Pedroza

4to Semestre; Matricula:189289

Jesús Francisco Caro

Cd. Obregón, Sonora. 18 de febrero de 2020

**Objetivos**

Los objetivos que cumplir en este proyecto es realizar cuatro *Shaders* distintos los cuales son: *Outline,* Entintado, Holograma (*Glitch*) y *Rim line*. Y combinarlos en dos *shaders* uno que tenga el *outline* y el entintado, mientras que el otro el holograma y el *Rim line*. Ambos *shader* finales serán colocados en un *sprite* para ver ambos efectos combinados con la opción de hacer ambas funciones al mismo tiempo.

**Introducción**

Para comenzar con el proyecto debemos dejar en claro algunos conceptos claros. Según la documentación de Unity, *Shaders* son scripts pequeños que contienen los cálculos de matemáticas y algoritmos para calcular el color de cada pixel renderizado, basándose en el input de iluminación y la configuración del *Material*. *Materials o materiales* son definiciones acerca de cómo la superficie debería ser renderizada, incluyendo referencias a texturas utilizadas, información del tiling (suelo de baldosas), tines de color y más. Las opciones disponibles para un material dependen en qué *shader* del material está utilizando. *Textures o texturas* son imágenes bitmap. Un material contiene referencias a texturas, para que el *shader* del Material puede utilizar las texturas mientras calcula el color de la superficie de un objeto.

Teniendo en cuenta los conceptos anteriores podemos inferir cómo funcionan entre sí a grandes rasgos. Mientras que los *shaders* les darán las distintas iluminaciones a las texturas a través del material, al material le asignaremos los *shaders* anteriormente mencionados para poder utilizarlos y colocarlos en cualquier superficie que queramos.

**Desarrollo**

Para crear los distintos *shaders* debemos de entender que es lo que hace cada uno de ellos y realizar una serie de pasos para poder crearlos y tiempo después combinarlos.

Todos los *shader* fueron un *shader sub grahp* para que así después poder combinarlos con su respectivo *shader* de manera más sencilla en un *shader 2D reder sprite unlit graph (experimental*). También comparten el mismo sprite.

El primer *shader* es el *outline* que será la silueta (la mayoría de las veces de color negro) que rodea a nuestro *spirte.* Para realizar el outline utilizamos el alpha de nuestro sprite para poder crear una silueta, en realidad usamos cuatro alphas cada uno por el punto cardinal correspondiente. Al sacarlos los alphas y sumarlos obtendremos la figura de nuestro sprite un poco más grande, esto es porque agregamos la silueta de nuestro sprite cuatro veces, pero un ligero movimiento hacia cada una de las direcciones. Después substraeremos el alpha original de nuestro sprite y el de la figura y el resultado lo multiplicaremos por el color que nosotros queramos tener en el resultado final. Esta multiplicación se la añadiremos a una capa de nuestro sprite y obtendremos ese efecto de outline que estábamos buscando.

El segundo *shader* es el entintado que será el que creara una nueva capa a nuestro *sprite* para darle color sin perder los colores de base que tiene nuestro *sprite*. Para realizar el entintado necesitamos dos colores base, uno que será el color original de nuestro sprite y otro color que será el color que este en una capa por encima del sprite. Este shader se utiliza para colorear el sprite sin modificar los colores originales del mismo. De manera general lo que se realiza asignarle un color base al sprite(blanco), a ese sprite le combinaremos el otro color que le queremos agregar, pero sin el alpha de ese color, ese alpha lo volveremos la opacidad de ese sprite, después crearemos otro sprite con el color que deseamos agregarlo y, por último, combinamos ambos, pero haciendo que nuestro segundo sprite ocupe únicamente el alpha y el sprite original ocupando los campos de RGB y obtendremos el efecto entintado.

El tercer *shader* es el holograma o efecto de *glitch* que le dará a nuestro *sprite* una esencia de estar incompleto y con fallas. Para realizar el holograma necesitamos un nodo voronoi, de manera resumida se puede decir que es un ruido el cual generado por un número pseudoaleatorio de células y un nodo tiling and offset el cual usaremos para deformar las células del voronio a lo largo y hacer que se repitan mediante un timer que colocaremos dentro del tiling and offset. Después agregaremos el voronoi al sprite mediante un multiply, luego lo combinaremos lo obtenido con el sprite original, pero haciendo que nuestro segundo sprite ocupe únicamente el alpha y el sprite original ocupando los campos de RGB, y así obtendríamos un ruido en el sprite. Le podemos agregar color al igual que noise sine wave que sirve para dar un efecto de encender y apagar.

Y, por último, el cuarto *shader* es el *rim line* que es como el *surface* del *sprite* el cual colorea los bordes.Para realizar el rim line necesitamos usar el nodo de fresnel effect al cual le ocupamos agregar un normal vector, un view direction y un vector 1 que funcione como su poder, entre más alto el valor menor es el efecto de rim line. Y, por último, se agrega color con un add.

Para poder realizarlo debemos primero descargar un *sprite* y colocarlo dentro de nuestro proyecto de unity, después en el archivo de unity crearemos un nuevo shader, en este caso un *shader sub grahp* para así llamar después dentro de otro *shader* el *outline* usándolo como propiedad y poder combinarlo con otro.

**Conclusión**

Podemos concluir que al utilizar los sub graph perdemos funcionalidades que si pudiéramos utilizar en un *shader 2D reder sprite unlit graph (experimental) o en PBR*

*Referencia:*

*Unity. (2019). UnityUser Manual (2019.3). febrero 18, 2020, de Unity Sitio web:* [*https://docs.unity3d.com/Manual/*](https://docs.unity3d.com/Manual/)

*Nintendo. (2020). La historia de Nintendo. febrero 18, 2020, de Nintendo Sitio web:* [*https://www.nintendo.es/Empresa/La-historia-de-Nintendo/La-historia-de-Nintendo-625945.html*](https://www.nintendo.es/Empresa/La-historia-de-Nintendo/La-historia-de-Nintendo-625945.html)

McCool, M. D., Qin, Z., & Popa, T. S. (2002, September). Shader metaprogramming. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (pp. 57-68). Eurographics Association.

McCool, M., Du Toit, S., Popa, T., Chan, B., & Moule, K. (2004). Shader algebra. In *ACM SIGGRAPH 2004 Papers* (pp. 787-795).

Blackman, S., Laliberte, P., & Reed, R. (2011). *Beginning 3D game development with Unity: the world's most widely used multi-platform game engine*. Apress.

Messaoudi, F., Simon, G., & Ksentini, A. (2015, December). Dissecting games engines: The case of Unity3D. In *2015 international workshop on network and systems support for games (NetGames)* (pp. 1-6). IEEE.

Hergaarden, M. (2011). Graphics shaders. *Graphics Shaders*, 1.

Prost, L. Molekuel-Rendering in Unity3D.

Evans, A., Romeo, M., Bahrehmand, A., Agenjo, J., & Blat, J. (2014). 3D graphics on the web: A survey. *Computers & graphics*, *41*, 43-61.

Schulz, H., Gotto, L., & Bartholdy, B. Visualization forms of graphic novels in virtual reality.

Doppioslash, C. (2018). How Shader Development Works. In *Physically Based Shader Development for Unity 2017* (pp. 3-16). Apress, Berkeley, CA.

Goldstone, W. (2009). *Unity game development essentials*. Packt Publishing Ltd.