Alumna: Blanca Lidia Castellanos Solis
Semestre: IV°
Carrera: IPM
Materia: Tópicos de Física
Instructor: Jesús Caro Cota

“Proyecto – Shaders Orientados a 2D”

Cd. Obregón, Sonora; 18 de febrero de 2020.

Existe una relación cercana entre Shaders, Texturas y Materiales en Unity.

- **Shaders:** son pequeños guiones que contienen los cálculos matemáticos y algoritmos para calcular el Color de cada píxel, el renderizado, basado en la entrada de iluminación y la configuración del material.
- **Texturas:** son imágenes de mapa de bits. Un material puede contener referencias a texturas, de modo que el sombreador del material pueda usar las texturas mientras calcula el color de la superficie de un GameObject.

  Además del Color básico (Albedo) de la superficie de un GameObject, las Texturas pueden representar muchos otros aspectos de la superficie de un Material, como su reflectividad o rugosidad.
- **Materiales:** se definen cómo se debe representar una superficie, incluyendo referencias a las texturas que usa, información de mosaico, matices de color y más. Las opciones disponibles para un material dependen del sombreador que esté utilizando el material.

Un material especifica un shader en ser utilizado, y el shader utilizado determina qué opciones están disponibles en el material. Un shader especifica una o más variables de textura que espera utilizar, y el inspector del Material de Unity le permite a usted asignar sus propios assets de textura a estas variables de textura.

Para la mayoría del renderizado normal - entendiendo por tal personajes, escenario, entorno, objetos sólidos y transparentes, superficies suaves y duras etc., el Standard Shader por lo usual es la mejor decisión. Este es un shader altamente personalizado el cual es capaz de renderizar muchos tipos de superficie en una manera bastante real.

Todas estas, 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 depende en qué shader del materia está utilizando.

Existe el **renderizado** 3D, el cual es el proceso de producción de una imagen basado en datos tridimensionales almacenados en tu computadora. También se le considera un proceso creativo, al igual que la fotografía o la cinematografía, ya que hace uso de la luz y produce finalmente imágenes.

Con el renderizado 3D, tus gráficos por computadora convierten modelos de wireframe 3D en imágenes 2D con efectos 3D fotorrealistas, o lo más cercano a la realidad. El tiempo de renderizado para una sola imagen o frame puede tomar desde unos cuantos segundos hasta incluso días. Existen dos tipos principales de renderizado en 3D y la diferencia más importante entre ellos es la velocidad a la que se calculan y procesan las imágenes: en tiempo real y sin conexión o prerrenderizado.

En el renderizado en tiempo real, que es lo más común en videojuegos o gráficos interactivos, las imágenes 3D se calculan a una velocidad muy alta, lo que permite dar la apariencia de que las escenas, que se componen de innumerables imágenes, se producen en tiempo real cuando los jugadores interactúan con tu juego.

Esa es la razón por la que la interactividad y la velocidad desempeñan un papel importante en el proceso de renderizado en tiempo real.

El objetivo principal es alcanzar el mayor grado posible de fotorrealismo a una velocidad de renderizado mínima aceptable, que normalmente es de 24 frames/segundo. Eso es lo mínimo que el ojo humano necesita para crear la ilusión de movimiento.

Aunque el renderizado se basa en una gran cantidad de cálculos sofisticados, los softwares modernos ofrecen algunos parámetros bastante sencillos de comprender y usar.

Al momento de realizar nuestro juego, el problema mas grande es probablemente seria la representación de las luces sobre cada una de las superficies,, siendo una de las tareas mas complejas que puede repercutir en el rendimiento de nuestro videojuego, en esta ocasión abordaremos el tema de las luces.

Existen varias luces en Unity, algunas de ellas son:

- **Point lights:**

Un punto de luz se encuentra en un punto en el espacio y envía luz en todas las direcciones por igual. La dirección de la luz que golpea una superficie es la línea desde el punto de contacto hasta el centro del objeto de luz. La intensidad disminuye con la distancia de la luz, llegando a cero en un rango específico.

- **Spot lights**

Al igual que una luz puntual, una luz puntual tiene una ubicación y un rango específicos sobre los cuales se cae la luz. Sin embargo, la luz puntual está limitada a un ángulo, lo que resulta en una región de iluminación en forma de cono. El centro del cono apunta en la dirección hacia adelante (Z) del objeto de luz.

- **Directional lights**

Las luces direccionales son muy útiles para crear efectos como la luz solar en sus escenas. Comportándose de muchas maneras como el sol, las luces direccionales pueden considerarse como fuentes de luz distantes que existen infinitamente lejos. Una luz direccional no tiene ninguna posición de fuente identificable, por lo que el objeto de luz se puede colocar en cualquier lugar de la escena.

- **Area lights**

Una luz de área se define por un rectángulo en el espacio. La luz se emite en todas las direcciones de manera uniforme a través de su área de superficie, pero solo desde un lado del rectángulo. No hay control manual para el alcance de una Luz de área, sin embargo, la intensidad disminuirá en el cuadrado inverso de la distancia a medida que se aleja de la fuente.

- **Emissive materials**

Al igual que las luces de área, los materiales emisivos emiten luz a través de su área de superficie. Contribuyen a la luz rebotada en su escena y las propiedades asociadas, como el color y la intensidad, se pueden cambiar durante el juego.

- **Ambient light**

La luz ambiental es la luz que está presente en toda la escena y no proviene de ningún objeto fuente específico. Puede ser un contribuyente importante para el aspecto general y el brillo de una escena.

Desde que comenzó la “revolución 3D” en el ámbito de los juegos de computadora, allá por mediados de la década de los 90’, la tendencia de la tecnología aplicada a este rubro ha sido trasladar el trabajo de procesamiento de gráficos tridimensionales, desde la CPU hacia la tarjeta de video.

El gran cambio se dio a partir de la incorporación de los Píxel shaders y Vertex shaders. Esto permitió a los programadores una mayor libertad a la hora de diseñar gráficos en tres dimensiones, ya que puede tratarse a cada píxel y cada vértice por separado. De esta manera, los efectos especiales y de iluminación puede crearse mucho más detalladamente, sucediendo lo mismo con la geometría de los objetos.

Shaders de Vértices

La responsabilidad de los Shaders de Vértices es asignar un valor a una variable especial gl_Position para crear los valores del espacio de trabajo (valores entre -1 y +1) en toda la zona del canvas. En nuestro Shader de Vértices de abajo estamos recibiendo valores de un atributo que definiremos como aVertexPosition. Estamos entonces multiplicando esa posición por dos matrices 4x4 que definimos como uProjectionMatrix y uModelMatrix e igualando gl_Position al resultado.

Por otro lado, un **Pixel Shader** no interviene en el proceso de la definición del "esqueleto" de la escena (conjunto de vértice, wireframe), sino que forma parte de la segunda etapa: la rasterización o render (paso a 2D del universo 3D). Allí es donde se aplican las texturas y se tratan los píxeles que forman parte de ellas. Básicamente, un Pixel Shader especifica el color de un píxel. Este tratamiento individual de los píxeles permite que se realicen cálculos principalmente relacionados con la iluminación en tiempo real, con la posibilidad de iluminar cada pixel por separado. Así es como se lograron crear los fabulosos efectos de este estilo que se pueden apreciar en videojuegos como Doom 3, Far Cry y Half Life 2, por mencionar sólo los más conocidos. La particularidad de los píxel shaders es que, a diferencia de los vertex shaders, requieren de un soporte de hardware compatible. En otras palabras, un juego programado para hacer uso de píxel shaders requiere si o si de una tarjeta de video con capacidad para manipularlos.

Referencias Bibliográficas

1. Lammers, K. (2015). Unity shaders and effects cookbook. Packt Publishing Ltd.
2. Pranckevičius, A., & Dude, R. (2018). Physically based shading in Unity. In Game Developer's Conference.
3. Ouazzani, I. (2017). Manual de creación de videojuego con Unity 3D (Master's thesis).
4. Zucconi, A., & Lammers, K. (2016). Unity 5. x Shaders and Effects Cookbook. Packt Publishing Ltd.
5. Dean, J. (2016). Mastering Unity Shaders and Effects. Packt Publishing Ltd.
6. Ciborro Montes, A. (2019). Renderizado de funciones de distancia mediante Raymarching en Unity.
7. Halladay, K. (2019). Writing Shaders in Unity. In Practical Shader Development (pp. 299-325). Apress, Berkeley, CA.
8. Iseki, F., Tate, A., Mizumaki, D., & Suzuki, K. (2017). OpenSimulator and Unity as a Shared Development Environment. Journal of Tokyo University of Information Sciences, 21(1).
9. Thorn, A., Doran, J. P., Zucconi, A., & Palacios, J. (2019). Complete Unity 2018 Game Development: Explore techniques to build 2D/3D applications using real-world examples. Packt Publishing Ltd.
10. Garrido Suárez, C. D. (2019). Sistema de posicionamiento y rotación 3D virtual (Doctoral dissertation).
11. Mateus Romero, F. (2015). Diseņo, desarrollo, modelado y animación de un juego con Unity3D.
12. Ferrando Monzón, G. (2016). GameBrush. Plugin de generación de formas abstractas y de alteración visual del entorno para Unity3D (Doctoral dissertation).
13. Doppioslash, C. (2018). Your First Unity Shader. In Physically Based Shader Development for Unity 2017 (pp. 17-32). Apress, Berkeley, CA.
14. Doppioslash, C. (2018). Your First Unity Lighting Shader. In Physically Based Shader Development for Unity 2017 (pp. 51-64). Apress, Berkeley, CA.
15. Doppioslash, C. (2018). How Shader Development Works. In Physically Based Shader Development for Unity 2017 (pp. 3-16). Apress, Berkeley, CA.

16. Alapont Granero, S. (2015). Scares for sale: diseño y desarrollo de un videojuego en Unity 3D. Audio e introducción a Leap Motion (Doctoral dissertation).
17. Unity . (2015). Experimental 2D Lights and Shader Graph support in LWRP. 18 de febrero de 2020, Sitio web: https://forum.unity.com/threads/experimental-2dlights-and-shader-graph-support-in-lwrp.683623/
18. (2019).Physically Based Rendering Material Validator. 18 de febrero de 2020, Sitio web: https://docs.unity3d.com/Manual/MaterialValidator.html
19. Unity . (2018).Texturas 2D. 18 de febrero de 2020, Sitio web: https://docs.unity3d.com/es/2018.4/Manual/Textures.html
20. Unity . (2019). Mathf.Clamp. 18 de febrero de 2020, Sitio web: https://docs.unity3d.com/ScriptReference/Mathf.Clamp.html