

Universidad La Salle Noroeste

**Tópicos De Física**

**Mapa Conceptual**

**Raúl Enrique Campbell Hidalgo (189152)**

**Ing. En Producción Multimedia**

Cd. Obregón, Sonora, 18 de February de 2020

**Historia**

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.

En primer lugar fue el filtro de las texturas, para lo cual se crearon chips especialmente dedicados para realizar esta tarea. Así nacieron las famosas placas aceleradoras 3D, que incorporaban dichos chips y una cantidad de memoria propia en la misma tarjeta. Luego, con la salida del GeForce 256 de NVIDIA, el procesador gráfico pasó a encargarse de lo que, hasta ese momento, realizaba la CPU. Estamos hablando de la función de Transformación e Iluminación (Transform & Lighting), utilizada para llevar a cabo los cálculos de geometría y de iluminación general de una escena en 3D. Hubo una versión mejorada de este motor, a la que se llamó de Segunda Generación. Ésta vino incluida a partir de la GeForce 2 y la gama Radeon de ATI, avanzando un poco más en cuanto a materia gráfica.

Un sombreador​ (llamado shader en inglés) es un programa informático que realiza cálculos gráficos escrito en un lenguaje de sombreado que se puede compilar independientemente. Es una tecnología que ha experimentado una rápida evolución destinada a proporcionar al programador una interacción con la unidad de procesamiento gráfico (GPU) hasta ahora imposible. Los sombreadores son utilizados para realizar transformaciones de vértices o coloreado de píxeles, entre otras labores, con el propósito de crear efectos especiales, como iluminación, fuego o niebla.

Para su programación, los sombreadores utilizan lenguajes específicos de alto nivel independientes del hardware sobre el que se ejecuten. Existen sombreadores de diversos tipos dentro de la arquitectura de una GPU, como por ejemplo sombreadores de vértices, sombreadores de geometría, y sombreadores de cómputo para realizar cálculos que no son gráficos, entre otros.

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.

**Tipos de Shaders**

Un **Vertex Shader** es “una función programable en tarjetas gráficas que ofrece flexibilidad al programador para renderizar una imagen. El sombreador de vértices se utiliza para transformar los atributos de los vértices (puntos de un triángulo) como el color, la textura, la posición y la dirección del espacio de color original al espacio de visualización. Permite que los objetos originales se distorsionen o cambien de forma de cualquier manera.”

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.

Los **geometry shaders** (o sombreadores de geometría) son un tipo de sombreador relativamente nuevo, introducido en Direct3D 10 y OpenGL 3.2; anteriormente disponible en OpenGL 2.0+ con el uso de extensiones. Este tipo de sombreador puede generar nuevas primitivas gráficas, como puntos, líneas y triángulos, a partir de las primitivas que se enviaron al comienzo de la tubería gráfica.

Los programas de sombreado de geometría se ejecutan después de sombreadores de vértices. Toman como entrada toda una primitiva, posiblemente con información de adyacencia. Por ejemplo, cuando se opera en triángulos, los tres vértices son la entrada del sombreador de geometría. El sombreador puede emitir cero o más primitivas, que se rasterizan y sus fragmentos finalmente pasan a un sombreador de píxeles.

Los usos típicos de un sombreador de geometría incluyen generación de sprites de punto, teselación de geometría, extrusión de volumen de sombra y renderizado de un solo paso a un mapa de cubos. Un ejemplo típico del mundo real de los beneficios de los sombreadores de geometría sería la modificación automática de la complejidad de la malla. Una serie de franjas de líneas que representan puntos de control para una curva se pasan al sombreador de geometría y, según la complejidad requerida, el sombreador puede generar automáticamente líneas adicionales, cada una de las cuales proporciona una mejor aproximación de una curva.

**Lenguajes para Shaders**

Existen basicamente tres lenguajes de alto nivel para programar Shaders, todos ellos utilizan un lenguaje tipo C y, en general, sus diferencias son mínimas.

**HLSL-DirectX High-Level Shader Language:** Posiblemente este sea uno de los lenguajes de shading más utilizados en la industria debido a la gran presión ejercida por [Microsoft](https://www.ecured.cu/Microsoft), combinado con el hecho que este fue el primero con estilo C utilizado para render en tiempo real. El HLSL se introdujo con [DirectX](https://www.ecured.cu/DirectX" \o "DirectX) 9.0, apareciendo antes que su competidor, el GLSL, además sus características han sido extendidas a lo largo de varias revisiones para poder competir con el GLSL.

**GLSL - OpenGL Shading Language:**También conocido como glslang, este lenguaje de alto nivel apareció con la llegada de OpenGL 2.0. El lenguaje ofrece un rico set de instrucciones desde sus inicios, unificando el procesado de vértices y fragmentos (pixels), y permitiendo 'loops' condicionales con estructuras en árbol. Históricamente, GLSL ha venido precedido por varias extensiones OpenGL como son ARB\_vertex\_program y ARB\_fragment\_program entre otras. Estas operaban con programas de bajo nivel escritos en ensamblador y con varias limitaciones. Estos lenguajes ensamblador ya han dejado de utilizarse para dar paso al GLSL. A partir de la especificación OpenGL 2.0, OGLSL pasa a formar parte del estándar. Esto implica que si una tarjeta es compatible con ésta versión, no hará falta el uso de extensiones, y garantizará su correcto funcionamiento.

**Cg - 'C for graphics' programming language:** Este lenguaje fue desarrollado por NVIDIA y ha sido desarrollado con el propósito de una fácil y eficiente integración en el proceso de producción. El lenguaje es independiente de la API e incluye una gran variedad de herramientas de soporte. Las primeras implementaciones de Cg eran bastante restrictivas debido a la abstracción de hardware pero a lo largo de varias versiones, se ha consolidado como un potente lenguaje de shading de alto nivel capaz de competir con el HLSL y el GLSL. Por otro lado, la sintaxis es prácticamente idéntica al HLSL y gracias a sus perfiles de compilación, puede generar código tanto para los diferentes Shader Models de DirectX como también para las diferentes extensiones OpenGL. En su última versión, la 1.5, incluso puede generar código GLSL.

**Actualidad del desarrollo de Shaders**

Ha habido una evolución lógica de los Shaders en estos últimos años. Este progreso tiene que ver, principalmente, con cuestiones internas de programación. Parámetros como la cantidad de registros disponibles, el número de instrucciones permitidas por programa y la incorporación de instrucciones aritméticas más complejas, entre otros, aumentando la flexibilidad a la hora de programarlos. Tanto OpenGL como DirectX han utilizado sus propios mecanismos de revisión y puesta al día de sus sistemas de shading. Mientras OpenGL se ha basando en el uso de extensiones que se han ido ampliando, DirectX ha definido diferentes modelos de shading (Shader Models), cada uno de estos modelos permite la realización de nuevos efectos y un mayor aprovechamiento del hardware.

**Aplicación en el Proyecto**

**Bibliografía**

***Editorwp\_6720, (2018, 28 marzo). Pixel Shaders y Vertex Shaders. Recuperado 17 febrero, 2020, de*** [***https://www.neoteo.com/pixel-shaders-y-vertex-shaders***](https://www.neoteo.com/pixel-shaders-y-vertex-shaders)

***Davis, Z. (s.f.). Definition of vertex shader. Recuperado 17 febrero, 2020, de*** [***https://www.pcmag.com/encyclopedia/term/vertex-shader***](https://www.pcmag.com/encyclopedia/term/vertex-shader)

***De Sousa, A., & Do Nascimiento, A. (2017). Beyond Programmable Shading.***

***Cardona Cabrera, J. A., & Pérez Barba, I. F. (2015). Desarrollo de un framework de programación para integrar el uso de shaders programables en un curso introductorio de graficación. Licenciatura en Ingeniería en Sistemas Computacionales.***

***Lewis, R. R. (2015, May). Making shaders more physically plausible. In Computer Graphics Forum (Vol. 13, No. 2, pp. 109-120). Edinburgh, UK: Blackwell Science Ltd.***

***Wylie, B., Moreland, K., Fisk, L. A., & Crossno, P. (2015, October). Tetrahedral projection using vertex shaders. In Symposium on Volume Visualization and Graphics, 2002. Proceedings. IEEE/ACM SIGGRAPH (pp. 7-12). IEEE.***

***Boyd, C. N., & Toelle, M. A. (2004). U.S. Patent No. 6,819,325. Washington, DC: U.S. Patent and Trademark Office.***

***Mitchell, J., McTaggart, G., & Green, C. (2006). Shading in valve's source engine. In Acm siggraph 2006 courses (pp. 129-142).***

***Lammers, K. (2013). Unity shaders and effects cookbook. Packt Publishing Ltd.***

***Zucconi, A., & Lammers, K. (2016). Unity 5. x Shaders and Effects Cookbook. Packt Publishing Ltd.***

***Pranckevičius, A., & Dude, R. (2014). Physically based shading in Unity. In Game Developer's Conference.***

***McDermott, W. (2012). Creating 3D Game Art for the iPhone with Unity: Featuring modo and Blender pipelines. Routledge.***

***Ouazzani, I. (2012). Manual de creación de videojuego con Unity 3D (Master's thesis).***

***Documentation, U. (2016). Materials, Shaders & Textures. en línia].[Últim accés: 26 de Febrer de 2019]. Disponible a: https://docs. unity3d. com/Manual/Shaders. html.***

***Dean, J. (2016). Mastering Unity Shaders and Effects. Packt Publishing Ltd.***

***Zucconi, A., & Lammers, K. (2016). Unity 5. x Shaders and Effects Cookbook. Packt Publishing Ltd.***

***Doppioslash, C. (2017). Physically Based Shader Development for Unity 2017: Develop Custom Lighting Systems. Apress.***

***De Byl, P. (2019). Holistic Game Development with Unity 3e: An All-in-One Guide to Implementing Game Mechanics, Art, Design and Programming. CRC Press.***

***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.***

***Doppioslash, C. (2018). Physically Based Shader Development for Unity 2017.***