Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

Implementación en Verilog de Unidad de Generacion de Rayos para GPU Theia

Por:

Josué David Vargas Amador

Ciudad Universitaria "Rodrigo Facio", Costa Rica

Diciembre de 2015

Implementación en Verilog de Unidad de Generacion de Rayos para GPU Theia

Por:

Josué David Vargas Amador

IE-0499 Proyecto eléctrico

Aprobado por el Tribunal:

MSc. Diego Valverde Garro
Profesor guía

MSc. Carlos Duarte Martínez
Profesor lector

MSc. Rodolfo Brenes Fernández
Profesor lector

Índice general

Índice de figuras

Índice de tablas

1 Introducción

1.1 Justificación

Los sistemas computacionales actuales poseen, dentro de su arquitectura, módulos de hardware especializados llamados Unidades de Procesamiento Gráfico (GPU, por sus siglas en inglés) encargados de acelerar el proceso de representación de objetos tridimensionales en la pantalla del computador.

Las unidades de procesamiento gráfico permiten la visualización de objetos mediante el cálculo de las primitivas que conforman el modelo abstracto de las imágenes. Las GPU implementan distintos algoritmos de representación gráfica, entre estos, uno es el algoritmo de Ray Casting.

El algoritmo de Ray Casting genera vectores (rayos) normalizados desde la perspectiva del usuario y calcula la intersección de los rayos con los objetos del escenario, además colabora con la formación de los colores, con la finalidad de crear las imágenes mostradas en pantalla.

Entonces los cálculos para la representación de objetos visuales en una GPU de tipo raycasting requieren de una arquitectura interna capaz de la generación de vectores (rayos) normalizados, para luego usar estas estructuras de datos en los módulos dedicados a la intersección de rayos. La generación de rayos requiere de instrucciones capaces de realizar cálculos aritméticos como multiplicaciones, sumas y restas, así como operaciones especializadas que permitan aproximar los valores de raíces cuadradas, por lo que el diseño lógico de una unidad dedicada facilitaría el proceso de creación rayos y permitiría añadir flexibilidad y modularidad al diseño de todo el GPU. Existen proyectos de hardware relacionados al diseño de arquitecturas de trazado de rayos que implementan unidades de generación de rayos propias como SaarCor de la Universidad de Saarland (Schmittler et al. (2004)) y RayCore de la Universidad de Sejong (Nah et al. (2014)).

En el caso de la GPU de tipo raycasting Theia, las especificaciones arquitectónicas indican la necesidad de una Unidad de Generación de Rayos (RGU, por sus siglas en inglés). La RGU debe poseer un conjunto de instrucciones necesarias para el cálculo de la normalización de vectores tridimensionales empleados en las siguientes etapas de funcionamiento del GPU.

1.2 Alcances del proyecto

La GPU Theia se encuentra en su tercera iteración, y en esta etapa tiene dos módulos principales dentro de su descripción de RTL en el lenguaje Verilog: la unidad de generación de rayos normalizados (RGU) y el módulo de intersección de rayos de tipo AABB (siglas en inglés de Axis Aligned Bounding Boxes).

El módulo de la RGU es un módulo que posee dentro de su descripción las instrucciones necesarias para el funcionamiento apropiado de la generación de rayos normalizados. Estas instrucciones deben ser capaces de proveer la información necesaria para programar el módulo de la RGU de manera que permita la normalización de los vectores mediante cálculo aproximado del inverso de la raíz cuadrada empleando el método iterativo para aproximación de raíces Newton-Raphson.

Posterior a esto se debe plantear un ambiente de verificación funcional que permita afirmar que el módulo RGU está cumpliendo con su papel dentro de la arquitectura y que puede generar la información requerida por los módulos de intersección de rayos.

1.3 Objetivos

Objetivo general

Desarrollar el modelo por comportamiento en Verilog de una Unidad de Generación de Rayos de un GPU tipo ray casting.

Objetivos específicos

Desarrollar el modelo por comportamiento en Verilog de una Unidad de Generación de Rayos de un GPU tipo ray casting.

- Investigar bibliografía sobre el mecanismo generación de rayos.
- Definir el mecanismo de generación de rayos normalizados en el GPU.
- Verificar el comportamiento funcional de la Unidad de Generación de Rayos en el GPU.

1.4 Metodología

1. Se procederá a investigar los conceptos fundamentales de la arquitectura de la GPU, el algoritmo de ray casting y sobre los posibles mecanismos de la generación de rayos normalizados.

2. Se buscará la implementación final de la arquitectura interna de la RGU de modo que contenga las instrucciones necesarias para la normalización.

3. Se simulará la ejecución del código en la RGU para generar los rayos normalizados necesitados por los módulos de intersección de rayos de tipo AABB.

4. Se verificará el comportamiento funcional del módulo RGU con la finalidad de establecer un marco de referencia para la futura validación del resto de la versión actual del GPU Theia.

1.5 Desarrollo

Este proyecto se estructura por medio de capítulos, cada uno tiene como tarea aclarar los siguientes tópicos:

1. Capítulo I: Introducción. Muestra la justificación del proyecto, los alcances y limitaciones, los objetivos y la metodología que permite cumplir los mismos.

2. Capítulo II: Antecedentes y Marco Teórico. Introduce al lector conceptos claves de arquitectura de unidades de procesamiento gráfico, algoritmo de raycasting, y plantea los casos de proyectos donde se han implementado chips.

3. Capítulo III: Implementación final de la unidad de generación de rayos. Aquí se describe la arquitectura final de la unidad, así como el método empleado usando las instrucciones de ésta para implementar la unidad.

4. Capítulo IV: Prueba de verificación funcional. Se comprueba la funcionalidad del módulo conductual de lenguaje Verilog por medio de un ambiente de verificación apropiado.

5. Capítulo V: Conclusiones y recomendaciones. Se muestran posibles resultados del proyecto y reflexiones sobre el futuro del proyecto.

2 Marco Teórico

2.1 GPU

Definición

Las unidades de procesamiento gráfico se encargan de rápidamente renderizar (representar) objetos 3D en forma de píxeles en la pantalla de la computadora típicamente por medio de arquitecturas de hardware basadas en la técnica de rasterización. La mayor parte de las GPU han sido diseñadas para realizar operaciones fijas organizadas en forma de pipeline para ir pasando vértices y píxeles a través de distintas etapas.

A continuación se mencionan las etapas principales del pipeline de gráficos:

1. El programa de usuario proporciona los datos al GPU en la forma de primitivas como puntos, líneas y polígonos que describen la geometría 3D.

2. Etapa geométrica: las primitivas geométricas son procesadas en base a los vértices y son transformados de coordenadas 3D a triángulos 2D en la pantalla..

3. Etapa de rasterización: en esta etapa se dibuja una imagen mediante el uso de los datos anteriormente generados así como de los cálculos computacionales por píxel. La salida es un conjunto de píxeles donde cada píxel posee sus propios atributos.

Los conjuntos de datos muy grandes que deben ser visualizados en tres dimensiones normalmente son creados usando representaciones de superficies mediante el dibujo de primitivas geométricas que crean mallas poligonales, pero las técnicas convencionales de rasterización al usarse en el renderizarizado de datos volumétricos producen pérdidas en la visualización. Las técnicas de renderización de volumen tienen más información que los métodos de renderización por superficie pero poseen una mayor complejidad y mayores tiempos de renderización.

2.2 Raycasting

Se presentan detalles acerca del algoritmo de raycasting en el cual se basa el GPU Theia para su funcionamiento.

Definición

El algoritmo de raycasting funciona haciendo cálculos un píxel a la vez, y para cada píxel la tarea básica es encontrar el objeto que es observado en la posición correspondiente a ese píxel en la imagen. Se puede decir que cada píxel ve en una dirección distinta y cualquier objeto que es observado por un píxel debe intersectar el rayo proveniente desde el punto de vista de la cámara. El objeto esperado es aquel que es intersectado primero por el rayo más cercano a la cámara. Una vez que el objeto es encontrado, se emplea el punto de intersección, la superficie normal, y la otra información del objeto para definir el color de cada píxel.

Entonces se puede decir que un algoritmo de raycasting tiene tres partes básicas:

1. Generación de rayo: donde se calcula el origen y la dirección de cada rayo (vector) del píxel correspoendiente en la vista de la cámara.

2. Intersección de rayo: donde se determina el objeto más cercano en la intersección del rayo proveniente de la cámara.

3. Shading: donde se calcula el color del píxel basado en los resultados de la intersección de rayos.

Con el objetivo de generar rayos, primero se necesita una representación matemática de un rayo. Un rayo en realidad es solo un punto de origen y una dirección propagación, una línea paramétrica en 3D que va desde el ojo llamado punto e hasta otro punto s que está en el plano de la imagen está dada por:

$$p(t) = e + t(s - e) \tag{2.1}$$

Esta fórmula implica que se empieza en el punto e y se avanza a través del vector s-e hasta llegar al punto p. Valores negativos de t implica que se encuentra detrás del ojo.

Un seudocódigo sobre el algoritmo de raycasting es el siguiente:

1. For every pixel # para cada píxel

2. Construct a ray from the eye # construya un rayo

3. For every object in the scene # para cada objeto en la escena

4. Find intersection with the ray # encuentre la intersección

5. Keep if closest # Guarde el más cercano

En términos generales se puede afirmar que la técnica de raycasting evalúa el color de cada pixel en la imagen al disparar un rayo a través de la escena desde la posición del observador. Si el rayo golpea el volumen, el color del pixel es calculado muestreando los datos a lo largo del rayo en un número finito de posiciones en el volumen y combinando cada resultado en uno solo. Este método tiene una limitación al ejecutarse en los CPUs: para vólumenes de datos grandes el tiempo de renderización para una sola imagen es muy alto para visualización en tiempo real.

Transformaciones

2.3 Ejemplos de GPU con unidades de generación de rayos

SaarCOR

DRPU

RayCore

Vizard II

2.4 Arquitectura de Theia

2.5 Métodos de normalización

Goldschmidt

Newton-Raphson

SRT-Redundant

Non-Restoring

Wong-Goto

2.6 Verificación funcional

Bibliografía

Nah, J., KWON, H., Kim, D., Jeong, C., Park, J., HAN, T., Manocha, D., y Park, W. (2014). Raycore: A ray-tracing hardware architecture for mobile devices. *ACM Transactions on Graphics, Vol. 33*.

Schmittler, J., Woop, S., Wagner, D., Paul, W., y Slusallek, P. (2004). Realtime ray tracing of dynamic scenes on an fpga chip. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*.