POLITÉCNICA

ETSIT UPM

UNIVERSIDAD POLITECNICA DE MADRID
E.T.S. DE INGENIEROS DE TELECOMUNICACION

A C T A D E E X A M E N

Asignatura: PROYECTO FIN DE CARRERA

TÍTULO DEL PROYECTO: FPGA DESIGN AND IMPLEMENTATION OF CRITICAL STAGES OF A GRAPHICS PROCESSING UNIT.

APELLIDOS Y NOMBRE:	CALIFICACIÓN
ROYER DEL BARRIO, PABLO	MATRICULA HONOR 10p.

Tutor: D. PABLO ITUERO HERRERO

Madrid, a 11 de Junio de 2010

EL VOCAL PRIMERO	**LA PRESIDENTA**	**EL VOCAL SECRETARIO**

MIEMBROS DEL TRIBUNAL

Presidenta: Dª. Mª LUISA LÓPEZ VALLEJO

Vocal: D. JUAN ANTONIO LÓPEZ MARTÍN

Secretario: D. PABLO ITUERO HERRERO

Proyecto Fin de Carrera

Título: FPGA Design and Implementation of Critical Stages of a Graphics Processing Unit

Autor: Pablo Royer del Barrio

Tutor: Pablo Ituero Herrero

Tribunal Calificador

Presidente: Dª. Mª Luisa López Vallejo

Vocal: D. Juan Antonio López Martín

Secretario: D. Pablo Ituero Herrero

Fecha de lectura: 14 de Junio de 2010

Calificación: MATRÍCULA DE HONOR (10.0)

Resumen

Desde finales de los años 90, las tarjetas gráficas tienen cada vez más presencia en la industria electrónica, hasta tal punto que hoy en día es difícil imaginarse un ordenador o video-consola que no disponga de una. El elemento clave de estas tarjetas es la unidad de procesamiento gráfico, o GPU por sus siglas en inglés. Una GPU toma la descripción de una escena en tres dimensiones, compuesta por polígonos y sus vértices así como un punto de observación, y los va transformando hasta obtener una imagen en dos dimensiones que se muestra por pantalla.

Este proceso conlleva una gran carga computacional pero admite una cierta latencia por lo que la solución que implementan las GPUs es la de un pipeline gráfico muy profundo con varias de las etapas replicadas en paralelo aprovechando la independencia de los datos que procesan. Una de las etapas claves del pipeline es la que toma triángulos, el polígono más simple que se puede obtener de los vértices, para transformarlos en fragmentos. Los fragmentos son porciones de pantalla de tamaño inferior o igual a un pixel que en las condiciones adecuadas (que no haya otro fragmento que los tape) se mostrarán por pantalla contribuyendo a la imagen final.

Este proceso, conocido como rasterización, se compone de varias etapas: en primer lugar se toman los vértices, expresados en unas coordenadas arbitrarias para adaptarlos a la resolución de pantalla deseada, a lo que se conoce como adaptación de pantalla, o *screen mapping* en inglés. A continuación viene la etapa de preparación de triángulos o *triangle setup* en inglés, en la cual se toman los vértices de tres en tres para formar triángulos y se calculan algunos parámetros de dichos triángulos como pueden ser las ecuaciones de recta de sus lados, su superficie u otros valores necesarios para la etapa siguiente. Finalmente se exploran los triángulos comprobando con que píxeles de la pantalla se solapan, y generando un fragmento para cada uno de ellos. Esta etapa se denomina recorrido de triangulos o *triangle traversal* en inglés, para la cual existen diferentes alternativas dependiendo de qué píxeles y en qué orden se comprueban.

En este proyecto se ha descrito en VHDL e implementado en FPGA cada uno de los tres módulos explicados anteriormente, haciendo más hincapié en el de *triangle traversal* del que se han estudiado tres alternativas para compararlas en cuanto a área, frecuencia máxima y eficiencia. También se han implementado tres módulos de *triangle setup* distintos, cada uno de ellos proporcionando los parámetros necesarios a su módulo de *triangle traversal* correspondiente. El módulo de *screen mapping*, en cambio, es común a las tres alternativas.

Las principales aportaciones de este trabajo son:

- La comparación de la implementación hardware de diferentes algoritmos de *triangle traversal.*
- El estudio de la cuantificación de cada una de las señales implicadas.
- Un análisis de la eficiencia de cada uno de los algoritmos.

En el primer capítulo de este proyecto se explica el funcionamiento general del pipeline gráfico así como la evolución que ha tenido. A continuación se explican más detalladamente las etapas de rasterización en las que se ha centrado el proyecto. Los capítulos siguientes se dedican a cada una de las etapas que se han implementado, mostrando en primer lugar las operaciones que deben realizarse para después explicar como se ha decidido implementarlas. Finalmente se muestran y comparan los resultados de las implementaciones y se dan ideas para trabajos futuros.

Abstract

The video game and interactive entertainment industry is getting revenues of tens of billions of dollars and increasing every year. The heart of the technological developments that make possible to keep satisfied the high demanding users is the Graphics Processing Unit (GPU). Apart from these purposes, the computational power and programmability of today's graphics hardware can be applied to many areas like computer graphics, film rendering, physical simulation, and visualization.

GPUs implement basic graphical operations, called primitives, optimized for graphics processing. The 3D graphics computations are organized into a graphics pipeline that performs a series of computation stages to go from a 3D model to the pixels on a monitor. The GPU processes and transforms the input vertexes into screen-space geometry, which in turn is divided into pixel-sized fragments, in a process called rasterization, according to which pixels are covered by that geometry. Each fragment is then associated with a pixel position on the screen. Finally, the fragments are processed and assembled into an image made of pixels [16].

Figure 1 shows a conventional graphics pipeline, which contains around a dozen stages. The input vertex stream passes through a computation stage that transforms and computes some of the vertex attributes generating a stream of transformed vertexes. The stream of transformed vertexes is assembled into a stream of triangles, each triangle keeping the attributes of its three vertexes. After that, the stream of triangles may pass through a stage that performs a clipping test. Then each triangle passes through a rasterizer that generates a stream of *fragments*, discrete portions of the triangle surface that correspond to the pixels of the rendered image. Fragment attributes are derived from the triangle vertex attributes. This stream of fragments may pass through a number of stages performing a number of visibility tests (stencil, depth, alpha and scissor) that will remove non visible fragments and then will pass through a second computation stage. The fragment computation stage may modify the fragment attributes using additional information from n-dimensional arrays stored in memory (*textures*). Textures may not be accessed as stream. The stream of shaded fragments will, finally, update the frame-buffer[8].

Figure 1: Graphics pipeline.

The typical input to the pipeline is tens to hundreds of thousands of vertexes that define the geometry of the scene. The basic elements are made up of triangles in general, since triangles can be described with three vertexes, and only reside in one plane. Thus a key stage in the graphics pipeline is the one that maps each pixel to a given triangle. To accomplish this task triangle traversal algorithms are required, whose computation is responsible for a significant performance overhead due to the huge number of vertexes which are processed. The hardware implementation of the triangle traversal should be carefully analyzed in order to minimize the performance penalty incurred by this stage. This is the objective of the work we present here. Very few previous approaches have dealt with the hardware implementation of triangle traversal algorithms. In [7] a specific full-custom implementation for a tile-based triangle traversal algorithm is studied, but paying special attention to power minimization. Other works focus on the quantization impact on the quality of the resulting images [6].

The main contributions of this work are the following:

- To the best of our knowledge this is the first time that the hardware implementation of different triangle traversal algorithms has been analyzed.
- The quantization of each involved signal has been studied.
- An in-depth analysis of the implications of each algorithm concerning throughput and latency has been performed.

The results of this work about the triangle traversal module have been sent to the XXV Conference on Design of Circuits & Integrated Systems on a paper after the title *On the Hardware Implementation of Triangle Traversal Algorithms for Graphics Processing* which is, by the time this thesis is presented, subject to peer review.

The first chapter of this work shows an overview of how the pipeline works and how it has evolved through the years. The next chapter will explain in detail the rasterization process on which this work has focused. The following chapters show how each one of the modules have been implemented, explaining firstly the operations to be performed. At the end the implementation results are compared, and ideas for future work are suggested.

Keywords

Computer graphics, Rasterization, Triangle traversal, Triangle setup, Screen mapping, Graphics processing unit, FPGA, VHDL, Zig-zag, Bounding-box, Hilbert curve, Depth interpolation.

Contents