Efficient and Scalable 3D Rendering Architecture

Péter Szántó, Béla Fehér
Budapest University of Technology and Economics
Department of Measurements and Information Systems
szanto@mit.bme.hu, feher@mit.bme.hu

1. Introduction

As the demand for complex display systems increases, three dimensional graphics rendering becomes an important feature in embedded systems, such as handheld devices (PDAs, mobile phones), set top boxes and onboard computers. Generating 3D graphics typically requires dedicated hardware, as even the fastest desktop CPUs are too slow to achieve real-time performance with good quality, not to mention the slower embedded parts. The main goal of this research project is to create a scalable rendering architecture feasible for system on chip (SOC) applications.
Typical embedded systems have quite different possibilities compared to desktop computers. From the 3D rendering point of view, the main drawback is the much lower external memory bandwidth, which is shared between the different SOC units – an appropriate architecture, therefore, has to save memory bandwidth by using it as efficiently as possible. Scalability is also a substantial feature, as different embedded systems may have quite different performance requirements.

2. Basics of 3D rendering

Real-time 3D rendering is based on triangles: the surfaces of complex objects are approximated by a 3D triangle mesh ([1]). These meshes are defined in their local coordinate system, and then transformed into the 3D world according to their position and orientation, so that the camera gets into the origin looking into the positive Z direction.
Transformation is followed by rasterization. For every screen pixel, the visible triangle – which is closest to the camera at the given screen position – is determined. In hardware implementation this is exclusively done with the Z Buffer algorithm, which requires a per-pixel buffer for storing the minimum Z value of the already processed triangles ([1-3]). The per-pixel Z value of a new triangle is compared with this buffer value to determine if it is closer to the camera than any of the already processed ones.

3. Hardware architecture

Figure 1 shows the simplified block diagram of the proposed architecture. To save memory bandwidth, on-chip buffers (grey boxes) are used. As these buffers for the entire screen are too large to fit into on-chip memories, the screen is segmented into small rectangles, and rasterization happens segment by segment. Processing starts by determining the visible triangle for every segment pixel (HSR Unit, [5-6]), followed by the computation of the output color value (Shading Unit) – by doing this only for truly visible triangle pixels, a lot of unnecessary work can be eliminated. To relieve the CPU, transparent objects are handled entirely in hardware ([4]).
However, processing all triangles of the frame in all segments would greatly decrease efficiency, therefore before rasterization, the Segmenting Unit generates a list for every segment, containing the triangles which cover at least one pixel in the given segment.
As different applications can have very different average triangle size, to maximize efficiency, the size of the screen segment is configurable. By analyzing the rendered frames, and varying the segment size between frames, even adaptive optimization can be achieved.

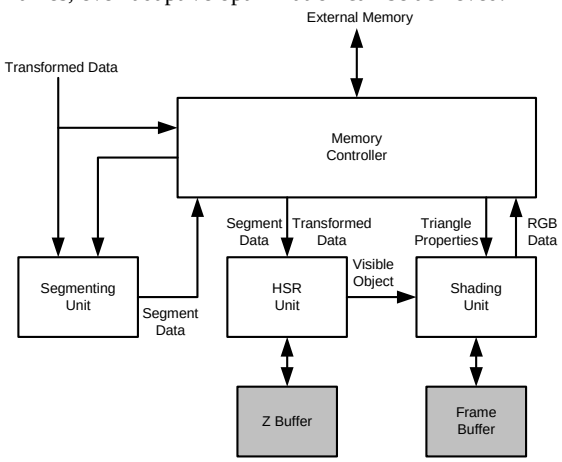

Figure 1. Rasterizer block diagram

3.1. Segmenting Unit

This processing unit works triangle by triangle. For every triangle, it steps through the affected segments, and generates the triangle list for the HSR Unit. The stepping algorithm is quite effective, as only those segments are evaluated which have at least one pixel covered by the processed triangle – so a new item in the list can be generated every clock cycle. In order to maximize reading performance in the HSR Unit, while at the same time minimizing memory size, the list is chained, and built up from 32-word blocks.

3.2. HSR Unit

The highly parallel HSR Unit is built up from several similar processing units. When it processes a triangle, all pixels of the segment are evaluated to decide which are covered by the triangle – with triangles covering only small fraction of the segment, this reduces efficiency; however it allows easier control mechanism and

deterministic processing time. The architecture is feasible to implement high performance edge anti-aliasing to improve quality on lower resolution screens.

3.3. Shading Unit

The Shading Unit computes the output color values for the segment. Basically, it is a programmable floating point ALU, which uses the attributes of the triangles. Such attributes are colors at the vertices, arrays assigned to the triangles and the shading program. With the appropriate compression of these attributes, memory bandwidth can be further reduced. The Shading Unit uses an on-chip Frame Buffer during processing, which is saved into the external memory after the segment is processed.

4. Results

At present, the Segmenting Unit and HSR Unit are implemented in Verilog HDL, achieving 100 MHz clock rate in our development Virtex2-6000 FPGA. The former unit can generate one output every clock cycle, while the HSR Unit is capable of processing two opaque pixels per clock.

Further research is necessary on the effect of adaptive segment size, effective data management and storage, high performance floating point ALU and effective compression schemes.

5. References

[1] A. Watt, 3D Computer Graphics, Addison-Wesley, 2000.

[2] R. N. Mahapatra, B. Murray: GEARS: Graphics Embedded Accelerated Rendering System, Technical Report TR-CS-2002-05-1

[3] H. Holten-Lund: Design for scalability in 3D computer graphics architectures. Ph.D. Thesis, Technical University of Denmark, 2001.

[4] P. Diefenbach, Pipeline Rendering: Interaction and Realism Through Hardware-Based Multi-Pass Rendering, Ph.D. Thesis, University of Pennsylvania, 1996.

[5] Kilgard, M. J., Everitt C. Optimized Stencil Shadow Volumes. Game Developer Conference, 2003.

[6] Y. Wang, S. Molnar, Second-Depth Shadow Mapping. Technical Report TR94-019, 1994.