Embedded 3D Graphics Core for FPGA-based System-on-Chip Applications

Hans Holten-Lund
Informatics and Mathematical Modelling
Technical University of Denmark
hahl@imm.dtu.dk

Abstract

This paper presents a 3D graphics accelerator core for an FPGA based system, and illustrates how to build a System-on-Chip containing a Xilinx MicroBlaze soft-core CPU and our 3D graphics accelerator core. The system is capable of running uClinux and hardware accelerated 3D graphics applications such as a VRML viewer.

The 3D graphics core is connected to a PLB 64-bit on-chip bus, and can render graphics into an on-chip tile buffer, which is later copied, using bus-master DMA transfers, to the frame-buffer in external DDR SDRAM memory. This memory is shared between the CPU, the 3D graphics core, and the video display which periodically reads from memory to display the final rendered graphics. The graphics core uses internal scratch-pad memory to reduce its external bandwidth requirement, this is achieved by implementing a tile-based rendering algorithm. Reduced external bandwidth means that the power consumption is reduced as well.

We show how an FPGA based embedded system is capable of most tasks in a single chip solution, without requiring additional CPU or graphics chips.

1. Introduction

Hardware accelerated 3D graphics is gaining influence in low-cost embedded devices such as GPS navigators, etc. For some applications FPGAs are on the way to replace CPUs, 3D graphics chips and other ASICs with soft-cores located inside the FPGA, giving a reconfigurable System-on-Chip solution. The main reason for this is the fact that FPGA chips are rapidly getting cheaper, while improving time-to-market. Although dedicated graphics ASICs are used for high performance applications such as game consoles, other applications which require moderate performance above that achievable with software rendering may be better served with a soft-core graphics processor in an FPGA. A similar trend can be observed with soft-core CPUs such as the Xilinx MicroBlaze and Altera Nios replacing traditional CPU chips by moving the CPU into the programmable FPGA fabric.

This paper presents a graphics core for such an FPGA based system, and illustrates how to build a simple SoC containing a Xilinx MicroBlaze soft-core CPU and our 3D graphics accelerator core. The system is capable of running uClinux and hardware accelerated 3D graphics applications such as a VRML viewer.

The target for this implementation is the Xilinx Virtex-4 ML401 XC4VLX25 FPGA evaluation platform. This board features 64 MB 32-bit external DDR SDRAM memory. The peak external memory bandwidth is 800 MBytes/sec when operating at 100 MHz. The board also provides an external ADV7125 3x8-bit RGB video DAC which is used for the VGA display.

The 3D graphics core is connected to a PLB 64-bit on-chip bus, and can render graphics into a frame-buffer using bus-master DMA to external DDR SDRAM memory. This memory is shared between the CPU, the 3D graphics core, and the video display which periodically reads from memory to display the final rendered graphics. The graphics core uses internal scratch-pad memory to reduce its external bandwidth requirement, this is achieved by implementing a tile-based rendering algorithm.

We show how an FPGA based embedded system is capable of most tasks in a single chip solution, without requiring additional CPU or graphics chips.

For the embedded application example with a GPS navigator, the GPS signal processing tasks can also be embedded in the FPGA to avoid the need for a dedicated GPS signal processing ASIC.

2. The Hybris Graphics Architecture

The Hybris graphics architecture [6] is scalable from a minimal embedded implementation to larger parallel implementations. The first FPGA implementation of the graphics architecture is presented in [7] where a simpler version of the core is implemented on a Celoxica RC1000PP PCI board using a Xilinx Virtex XCV1000 FPGA. The FPGA is streaming the input data from a host PC via the PCI bus, and outputs the rendered graphics directly to a VGA monitor.

In this context we will look at a new implementation suitable for an embedded System-on-Chip based on a cheap FPGA and low-cost 32-bit wide DDR SDRAM external memory. The FPGA contains an embedded MicroBlaze CPU and on-chip buses which replaces the PC from the earlier PCI-bus based implementation. To keep the cost low, the external memory is shared as main memory for the CPU, graphics memory for the 3D graphics accelerator and framebuffer memory for the display. This means that we need to have a bandwidth budget for the complete system. Internally in the FPGA all the main memory traffic is also present on a bandwidth matched 64-bit PLB (Processor Local Bus) on-chip bus.

The Hybris graphics architecture uses tile based rendering to allow for scalable parallel implementations. The method gets its name as it divides the screen into rectangular tiles. Each tile is relatively small so it can easily fit into on-chip memory next to the rendering engine. While we can divide the screen into many tiles, e.g. a 640x480 pixel framebuffer can be divided into 20x15 32x32 pixel tiles, we cannot afford to implement 20x15 tile rendering engines running in parallel. If we have one tile engine rendering to a virtual tile, we can render all tiles in sequence. This is also known as *virtual local framebuffer* see [5]. Similar techniques are also used in [8, 4, 9, 3] as well as by GigaPixel, which was acquired by 3dfx and then Nvidia. Some more recent work is [1, 2]. The advantages are that we do not need to access off-chip memory while rendering a tile, that we can use more bits per pixel in the small tile than we would in a global framebuffer without increasing off-chip bandwidth, that we can render multiple tiles in parallel, and that we can render at a higher resolution to implement supersampled anti-aliasing again without increasing off-chip bandwidth. Unfortunately since we must collect and bucket sort input data (triangles) for a tile before rendering it, we need a potentially large input buffer, which is the main drawback of the tile based rendering method. This is not a problem for scenes with a relatively low triangle count.

3. MicroBlaze Soft-Core CPU

The MicroBlaze [10] is a 32-bit soft-core RISC processor from Xilinx, optimized for FPGA implementation. We use the MicroBlaze CPU core mainly because it is well supported by the Xilinx implementation tools (EDK 7.1 and ISE 7.1), allowing us to focus our work on the system-design and hardware accelerator design. In addition to the usual on-chip bus-interfaces, MicroBlaze also provides eight Fast Simplex Links (FSL) which are useful for direct communication with hardware accelerators. For this application we use the MicroBlaze CPU to run the user programs and if necessary also the uClinux operating system.

The MicroBlaze CPU is also used for the front-end of the graphics architecture. This means that 3D transformation and lighting as well as triangle set-up is done in software on the CPU. The new MicroBlaze CPU version 4.0 provides hardware floating point support to help applications such as this, removing the need to convert the program to fixed-point. In a future implementation a hardware floating point vector co-processing module can be connected to the CPU using one of the FSL links. This makes it possible to implement dedicated matrix-vector multiplication and dot product hardware which is useful for accelerating 3D transformation and lighting. Since a FSL core does not need to send anything back to the CPU, the FSL core can be used to send its processed data directly to the graphics processor core or main memory, bypassing the overhead of sending it back through the CPU and its bus interface.

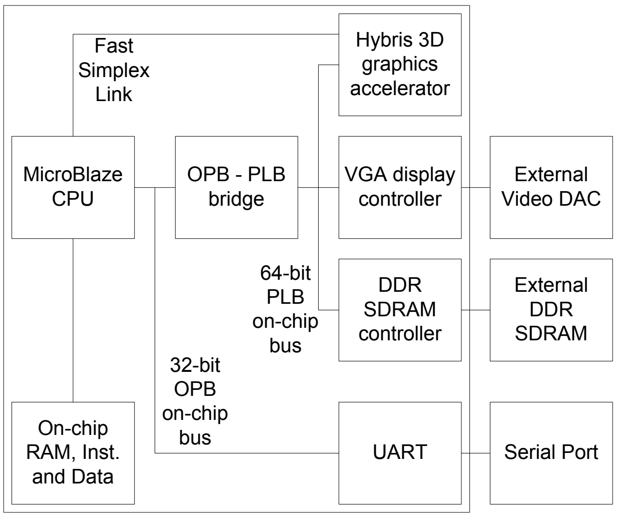


Figure 1. Overview of the embedded graphics system.

4. A SoC with CPU and Graphics Core

An overview of the System-on-Chip is presented in Figure 1. In addition to the graphics and memory subsystem the system contains the MicroBlaze CPU and two on-chip buses. We need to use the 64-bit PLB (Processor Local Bus) on-chip bus to support the performance requirements of the graphics system, while the 32-bit OPB (On-chip Peripheral Bus) is required for the MicroBlaze CPU. The on-chip buses are bridged via an OPB-to-PLB bus bridge which has a slave interface on the OPB side and a master interface on the PLB side. The bridge allows the CPU to initiate accesses to the devices on the PLB bus.

An important part of the PLB-side of the system is the main memory subsystem which is controlled by the DDR SDRAM controller. The DDR controller provides access to external DDR memory via the PLB-bus. The PLB bus bandwidth is dimensioned so it will match the bandwidth of the external memory. In practice this is accomplished by using 32-bit wide DDR SDRAM, which is represented internally as single data rate 64-bits on the PLB on-chip bus. If wider, e.g. 64-bit, DDR SDRAM is used the PLB on-chip bus may not be fast enough to match the memory bandwidth, as we cannot change the bus-width of the PLB, only the frequency. In such a case a dedicated link between the graphics core and the DDR memory controller core will be necessary. The DDR SDRAM in this system is built from two infineon 16Mbit x 16 chips which each internally uses four 8192 row by 256x32 column DRAM banks, forming 64 MBytes of main memory with a page size of 4 kBytes.

The other devices on the PLB bus are a VGA video display core and the 3D graphics accelerator itself. The VGA video display core is a simple framebuffer display system reading pixels from a framebuffer in main memory. It stores an entire scanline of pixels in on-chip memory which allows it to use a fixed pixel clock for sending pixel data to the external video DAC, while periodically reading from the framebuffer at the full bus-speed. The VGA video display core is a PLB bus-master since it initiates the read transactions for reading from the framebuffer.

The framebuffer memory area in the main memory can be written to by the CPU (through the bus bridge), or by the 3D graphics accelerator core. This makes it possible to combine software rendering running on the CPU with hardware rendered pixels from the 3D graphics core.

The 3D graphics core needs to read from a list of triangles to render, and also needs to write to the framebuffer. To minimize the bandwidth requirement on the PLB on-chip bus and external memory, we use an on-chip scratch-pad memory in the 3D graphics core. The scratch-pad memory is used for rendering pixels in a single e.g. 32 x 32 pixel tile, we will later find the optimal tile size. By rendering the pixels into a scratch-pad RAM, we can later use a block-transfer to write a completed tile into the framebuffer. This similar to the virtual local framebuffer scheme as described in [5], which is often known as tile-based rendering. To support tile-based rendering we must first sort all triangles to be rendered into buckets corresponding to the tiles they overlap, see figure 2. Bucket sorting is an additional step not found in most commercial rendering hardware because of its added complexity and increased per-triangle cost and the requirement of a memory buffer for storing the bucket sorted triangles (figure 3). This imposes an upper limit on the number of triangles we can process in a single pass. However we gain several advantage with the on-chip scratch-pad memory which can be used e.g. for cheap depth-buffering using only on-chip memory. We can also use a simple solution for anti-aliasing where a supersampled image is first rendered to the scratch-pad memory and then down-filtered when writing the resulting tile to the framebuffer. This way a cheap way to do anti-aliasing can be provided for low-resolution embedded displays to improve the visual quality of the graphics. With minor changes to the rendering algorithms it is also possible to use sparse supersampling instead of normal full supersampling, Sparse supersampling focuses on improving the visual quality by focusing the sample rate for nearly vertical and horizontal edges, which is where the human eye is most sensitive to aliasing artifacts. This can be achieved by using one of the sub-pixel sampling patterns in figure 4.

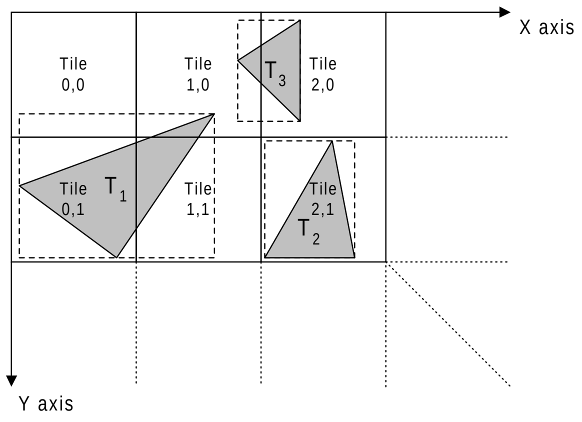


Figure 2. Examples of overlap in a tile-based renderer. Triangle T_2 is completely inside one tile. T_3 overlaps two tiles. T_1 overlaps 4 tiles if bounding box bucket sorting is used, but will only overlap 3 tiles if exact bucket sorting is used.

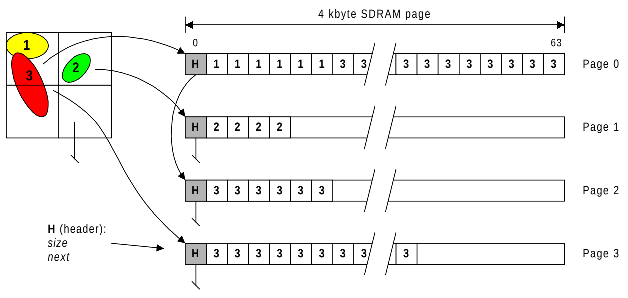


Figure 3. Memory management of triangle nodes in the bucket sorted triangle heap. Triangles are allocated in buffers of 63 triangles. A linked list of buffers are used, if space for more triangles are needed in a bucket. In this example the triangles of object 1 go to the first buffer, page 0. Object 2 is placed in page 1. The triangles of object 3 overlapping the upper left tile fills up page 0 and is continued in page 2. The triangles overlapping the lower left tile goes to page 3. The lower right tile is empty and nothing is allocated for it.

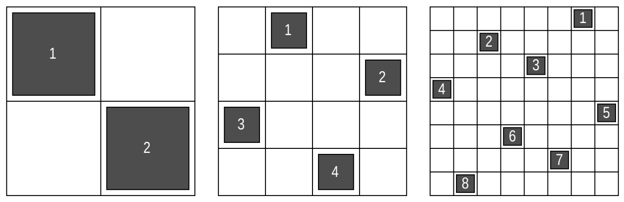


Figure 4. Sparse supersampling sub-pixel sample positions within a pixel. Left: 2 samples in a 2x2 grid. Middle: 4 samples in a 4x4 grid, Right: 8 samples in a 8x8 grid.

5. Optimizing the Hybris rendering engine

When building an embedded graphics system we must optimize the datapaths so their performance matches what is achievable given the limitations of the external main memory, internal on-chip scratch-pad memory, CPU caches and on-chip buses.

With a tile-based rendering engine one important parameter is the tile size. The size of a tile is mainly dictated by the available on-chip RAM resources, and also by the overlap factor which affects the overhead of the bucket sorting step. A smaller tile size reduces the demands of on-chip memory, but will also result in a larger overlap factor as the probability of a triangle overlapping multiple tiles will increase. A larger tile size will improve the memory coherence within the tile and will also help to reduce the overlap factor as a triangle will be less likely to overlap multiple tiles. Figure 2 shows a few cases of triangle-tile overlap.

This also depends on the triangle size distribution in the 3D scenes to be rendered. Assuming a typical triangle area of up to 64 pixels will keep the average overlap factor below 2 when using a tile size of 32x32 pixels. This is documented in [6].

Now that we know the bandwidth and memory limitations, we need to use this knowledge to properly configure the graphics architecture for a suitable implementation.

For the VGA video display we need to allocate main memory bandwidth for video refresh. With 640 x 480 32-bit pixels at 60 Hz and a 25 MHz pixel clock, the VGA display core will require an average bandwidth of 74 MBytes/sec from the main memory (100 MBytes/sec during active display and 0 during the blanking period). Updating the framebuffer at the same rate will require another 74 MBytes/sec.

The MicroBlaze CPU running at 100 MHz can at peak use up to 400 MBytes/sec bandwidth through the 32-bit OPB-to-PLB bus bridge. This is however not very likely, because the CPU is configured to use 16 kByte instruction- and data caches. The real bandwidth used by the CPU is difficult to predict, but is far below the peak figure mentioned before. If necessary, the CPU can also run (small) programs from internal block ram based memory.

This leaves a worst case main memory bandwidth of at least 800-400-100-74 = 226 MBytes/sec available for other use. We can use this for either a higher resolution framebuffer, or for the 3D graphics accelerator core.

The 3D graphics accelerator core will also need to read from a bucket sorted head of triangles from the main memory. The bandwidth required for this depends on the number of triangles in the scene, the size of a triangle description node, the triangle tile overlap factor and the frame rate.

Any remaining bandwidth can be used for other purposes such as a future implementation of texture mapping, or having a global depth (Z) buffer in main memory. Note that the tile-rendering algorithm does not require a global depth buffer. However if the number of triangles in a scene exceeds the memory allocated for the bucket sorted triangle heap, a multipass algorithm could be used to save the depth-buffer tiles to main-memory between the passes.

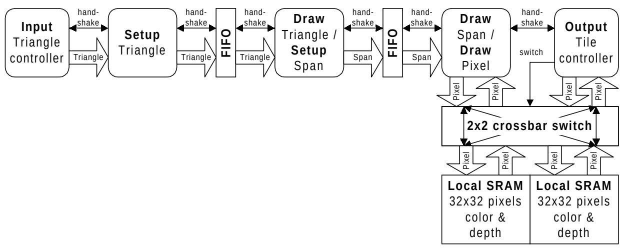


Figure 5. Architectural overview of the tile rendering engine back-end pipeline. FIFO buffers are placed between iterating pipeline stages to help average out load imbalances. The double buffered tile buffer allows the tile engine to render one tile while the previously rendered tile is being copied to the global framebuffer.

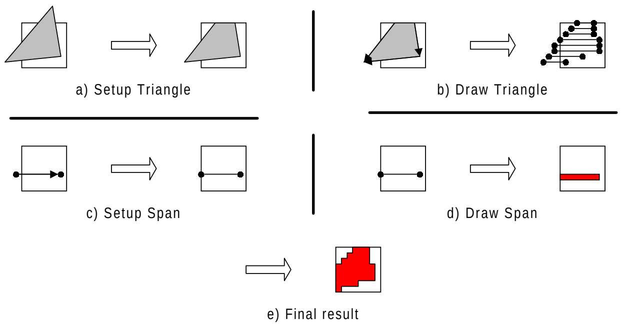
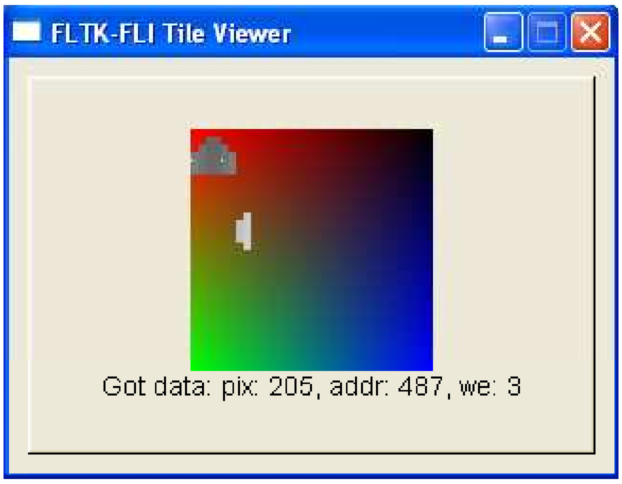


Figure 6. Processes in the tile rendering engine back-end pipeline. a) Setup Triangle adjusts y-parameters to fit tile. b) Draw Triangle iterates over the active scanlines, generating spans. c) Setup Span adjusts x-parameters to fit tile. d) Draw Span iterates over the active pixels in the span, performing per-pixel shading and depth testing. e) Final result for drawing one triangle.

5.1. Rendering core performance

The 3D graphics core in figure 5 was implemented as a core for the FPGA. Figure 6 shows how a triangle is rasterized by the tile rendering engine. The small FIFO buffers provide dynamic load balancing between the stages. The last pixel drawing stage is able to render a pixel on every clock cycle, provided that the previous stages can supply data fast enough. For this reason the previous stages should be designed so that the per-triangle cycle count matches the per-scanline (times the number of scanlines) cycle count and also the per-pixel cost (times the triangle area). If we only render relatively large triangles it



Got data: pix: 205, addr: 487, we: 3

Figure 7. Graphical testbench window used for verification of the tile rendering engine. The testbench was written in C using FLTK for graphics and connects to ModelSim using FLI (Foreign Language Interface). Each pixel in the 32x32 tile has been magnified to a 4x4 pixel block and the shaded background makes it easier to see both dark and bright pixels.

is sufficient to have a fast pixel drawing stage. Some applications require a large number of small triangles (this is the trend today for increasing detail), one example is medical visualization where we usually end up with millions of triangles in a reconstructed 3D surface. For such applications, we anticipate a large number of small triangles which places significant demands on per-triangle processing as well. As a result our final implementation is dimensioned so it is able to handle very small triangles without slowing the pixel drawing rate, i.e. the per-triangle and per-scanline processing time is also one clock cycle, i.e. we have single cycle triangle setup and scanline setup. If the triangle area is larger than a few pixels the tile rendering engine can be balanced to improve the pixel throughput relative to the per-triangle throughput, this can be done by reducing the hardware resources in the triangle and scanline stages by using a multicycle machine for computing the initial setup. For performance reasons another approach is to speed up the pixel processing speed instead, which can be done by using parallel pixel pipelines [6].

The rendering core has been tested in simulation using ModelSim and also synthesized and tested on the FPGA itself. A 32x32 pixel tile engine capable of rendering one one pixel triangle per clock cycle occupies 2107 slices and 3 RAMB16 in a Virtex4 LX25, and can operate at 70 MHz with single cycle triangle processing. The critical path here is in the triangle setup stage, providing fast performance for small triangles. If faster per-triangle processing speed is needed we can pipeline the triangle setup and the scanline setup stages. The design is well suited for automatic pipelining by adding several registers to the output of each stage and then synthesizing it with register balancing turned on in a synthesis tool which supports it. Depending on the triangle size distribution in the application we can also allow the triangle setup to take multiple cycles, either using a FSM controller/datapath or use a simpler setup with a multicycle combinational path.

Note that the above results are for 8 bits per pixel + 32 bit per pixel for the Z (depth) buffer used in the tile rendering algorithm, which is mapped to the 3 block rams (single buffered, 6 are needed in the double buffered version shown in figure 5). Note that the tile size can be adjusted to match the number of block rams available for the tile rendering engine, the desired number of bits per pixel and whether we need single or double buffering. The tile size also affects the overlap factor during bucket sorting of the triangles prior to rendering; larger tiles give lower overlap, but use more memory and makes load balancing worse in a parallel implementation. In [6] we show that a tile size of ca. 32x32 pixels is a good compromise. This has also been shown recently in [1].

For verification we used a specialized graphical testbench written in C, which interfaces to ModelSim SE through its FLI interface. This graphical testbench provides a quick visual overview of what is written to the tile pixel- and depth-buffers cycle by cycle. In practice this is done by writing a VHDL entity which has the relevant signals as input ports which then connects to the compiled testbench which was written in C and uses FLTK to display a graphics window. Figure 7 shows an example of the graphical testbench window. The synthesized tile rendering engine has also been verified to work in hardware using a VGA monitor to display data being read directly from the tile buffer, showing the same images as the graphical testbench. Figure 8 shows an example of the type of graphics that can be rendered with the graphics core.

6. Summary and future work

The presented graphics system can be tuned for performance depending on the target application. As a starting point we are able to rasterize small triangles at a high rate. Rendering one one pixel triangle per cycle at 70 MHz gives a peak triangle processing rate of 70 million triangles per second. This assumes a memory system fast enough to supply data to the tile engine. Internally each triangle is sent to the tile rendering engine on a 333 bit wide bus in a single cycle. This translates to a peak bandwidth of roughly 3 Gbytes/sec for reading the input data. The total bandwidth is only 800 MBytes/sec on the FPGA evaluation board, so we cannot stream data fast enough from the main memory. One way to solve this data memory bandwidth problem could be to store a small dataset in the FPGA block rams, although that would limit the usefulness of the system.

Note that the above bandwidth is a worst case calculation, if the graphics system is used to render more typical

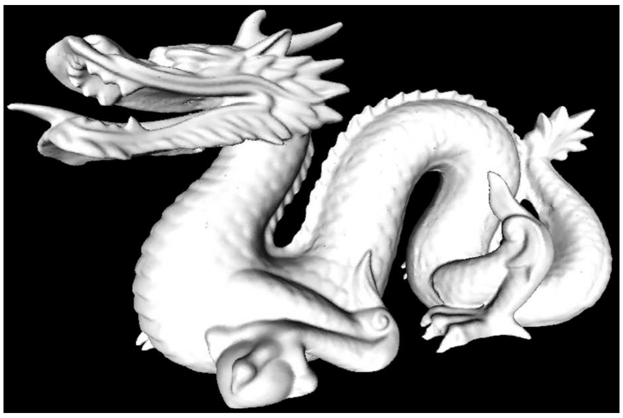


Figure 8. Example of an object rendered with the FPGA-based 3D graphics core. This is the Standford Dragon laserscan with 870000 triangles. The image was rendered in 2 seconds on the Virtex 1000 FPGA board running at 25 MHz.

scenes with larger triangles with an average area of about 64 pixels, the bandwidth requirement for the external triangle buffer will also drop by a factor 64, in the case of a tile rendering engine rendering one pixel per cycle. Triangles with an average area of 64 pixels also provide a good match for the 32x32 pixel tile size, giving an overlap factor of 2.

To fully utilize the speed potential in this graphics system, we need faster off-chip memory similar to the wide and fast DDR2 SDRAM used in modern graphics cards. Alternatively we can suggest that an evaluation board such as the ML401 is more suited for processing intensive rather than memory intensive applications. A Spartan 3E FPGA with wider and faster external RAM might be better suited for a low-cost implementation.

A complete System-on-Chip for an embedded graphics system will also need to do transformation, lighting and triangle setup before sending the triangles to the tile rendering engine for rasterization. While this can be handled in software on the MicroBlaze CPU version 4.0 with a hardware floating point unit, a dedicated transformation and lighting module can be implemented using e.g. the new Xilinx core-generator floating point cores. A dedicated vector processing unit will make it possible create a balanced graphics system that can both generate transformed and lit triangles, and also render them at the same rate. In the future is will be interesting to investigate the use of the PowerPC hard-core CPU found in the new Virtex-4 FX FPGAs, as it provides an Auxiliary Processor Unit interface to accelerator cores similar to the FSL links found in the MicroBlaze. However we must be careful not to build a system that requires an FPGA which is too expensive, which would make it difficult to compete against systems using dedicated ASICs.

References

[1] I. Antochi, B. Juurlink, and S. Vassiliadis. Selecting the optimal tile size for low-power tile-based rendering. In *Proceedings ProRISC 2002*, pages 1–6, November 2002.

[2] I. Antochi, B. Juurlink, S. Vassiliadis, and P. Liuha. Memory bandwidth requirements of tile-based rendering. In *Proceedings of the Third and Fourth International Workshops SAMOS 2003 and SAMOS 2004 (LNCS 3133)*, pages 323–332, July 2004.

[3] T.-C. Chiueh. Heresy: A virtual image-space 3d rasterization architecture. *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 69–77, August 1997.

[4] J. Eyles, S. Molnar, J. Poulton, T. Greer, A. Lastra, N. England, and L. Westover. PixelFlow: The realization. *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 57–68, August 1997.

[5] Foley, van Dam, Feiner, and Hughes. *Computer Graphics, Principles and Practice*. Addison-Wesley, second edition, 1990.

[6] H. Holten-Lund. *Design for Scalability in 3D Computer Graphics Architectures*. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, July 2001.

[7] H. Holten-Lund. FPGA-based 3D graphics processor with PCI-bus interface, an implementation case study. *NORCHIP 2002 Proceedings*, pages 316–321, November 2002.

[8] M. Kelley, S. Winner, and K. Gould. A scalable hardware render accelerator using a modified scanline algorithm. *SIGGRAPH Proceedings*, pages 241–248, July 1992.

[9] S. Nishimura and T. L. Kunii. Vc-1: A scalable graphics computer with virtual local frame buffers. *SIGGRAPH Proceedings*, pages 365–372, August 1996.

[10] Xilinx. *MicroBlaze Processor Reference Guide, UG081*, v5.1 edition, April 2005.