An Application Specific Reconfigurable Graphics Processor

Hans Holten-Lund
IMM, DTU
Overview

- Currently there is a trend for more programmability in graphics hardware.
- Where is the graphics industry headed?
- What about reconfigurable circuits?
  - This could be a comeback for “software” rendering, but not as we know it.
  - Parallel graphics systems?
- Demo:
  - A working graphics processor implemented in an FPGA.
Current trends

- **Past:**
  - Fixed function graphics processors.

- **Present:**
  - Programmable vertex and pixel processors.

- **Future:**
  - Reconfigurable vertex & pixel datapaths?
    - Modify instruction set of vertex & pixel processors?
  - Reconfigurable architecture?
    - Configurable parallelism?
    - Rasterization algorithm optimizations?
    - Support new graphics algorithms?
Why reconfigurable?

- Reconfigurable graphics processors could allow:
  - Specialized graphics primitives (e.g. subdivision surfaces)
  - Customized pixel processing (e.g. fragment sorting for correct multilayer transparency)
  - Avoiding software emulation (e.g. when you try to run a DX9 program on DX7-only hardware).

- New ways to process triangles and other primitives:
  - Current GPU’s use a fixed triangle processing pipeline.
  - The triangle processing part of the graphics processor should also be programmable, not just vertex processing.
  - Enables: Displacement mapping, procedural geometry, deformable models, collision detection, adaptive level-of-detail, soft shadows, physics simulation, etc…

- New exciting graphics algorithms appear all the time, we would like to have them running in hardware as well!
Reconfigurable architecture:

- Alternative rendering architectures:
- Tile-based rendering, where triangles are sorted according to screen regions.
  - Tiles can also be rendered in parallel allowing the graphics system performance to scale.
- Framebuffer organization could be more flexible:
  - How about a distributed framebuffer for large display walls?
  - Direct display of a floating-point framebuffer?
- Hardware Ray-Tracing.
  - For global illumination.
Reconfigurable hardware.

- Types of hardware:
  - Fixed-function ASIC (Application Specific IC)
    - E.g. older non-programmable hardwired graphics processors.
  - General purpose CPU (Pentium 4, etc.)
    - Fixed function but can run any program.
  - Application Specific Processors, ASP
    - Runs specialized code. (ex: Vertex shaders.)
  - Fixed-function ASIC with multiple embedded ASPs
    - More flexible, but still mostly hardwired.
  - Field Programmable Gate Array (FPGA)
    - Fully reconfigurable chip, can emulate any hardware.
Designing a custom graphics processor

- The task of designing a custom graphics processor from scratch is much like writing a software renderer.
  - In fact, you should start by writing it in C and test it thoroughly.
  - When the software model works, the design should be partitioned into hardware modules.
  - Or, we could use similar techniques to create a cache-optimized software renderer.
  - The C models are then translated to a HDL (Hardware Description Language) description in VHDL, Verilog or SystemC.
  - Using logic synthesis and place & route tools (a silicon compiler), the HDL code can be used to create a chip (ASIC) ready to be fabricated.
  - The same tools can be used to create a configuration pattern for an FPGA.
Why is an FPGA so interesting?

- It allows anybody to build graphics hardware! 😊
  - Tough learning curve.
- Reconfiguration:
  - In a few seconds the entire chip can be reconfigured, by loading a new configuration bit-pattern.
  - Partial self-reconfiguration in a running FPGA is also possible.
- Other interesting uses:
  - An FPGA is used in the European Mars Express Lander “British Beagle 2”.
    - To save weight and power, the FPGA can only control a single instrument at a time, but reconfigures itself on demand when another instrument is used. Memory chips for storing multiple configurations are lighter than having all the hardware in place at once.
  - Emulation of old hardware: Original Pac-Man game on an FPGA.
  - Real-time verification of large ASICs using multiple FPGAs.
Practical issues with FPGAs:

- **Cost issues:**
  - At low volume, very low cost compared to ASICs.
  - At high volume, expensive!

- **Speed issues:**
  - FPGAs are rapidly catching up with standard-cell based ASICs, there is only a factor 5 in clock speed difference today. The gap is getting smaller for each generation.

- **Area issues:**
  - Major concern, as most of the FPGA’s chip-area is used for the reconfiguration network. => low silicon area utilization.
  - ASIC based graphics processors are 100+ Mtransistor designs using many floating point units, etc.
Do we really need to use *programmable* vertex & pixel processors in an FPGA based GPU?

- We should! - The size of a pipelined RISC style processor is quite small in a modern FPGA.
- A pipelined dedicated T&L datapath is still possible, and will be faster but also larger.
- In the FPGA, programmable vertex & pixel processors can be customized to suit the specific vertex & pixel programs used in the application.
A Scalable Graphics Architecture

Some reasons for why we should look at scalable graphics architectures:

- Since an FPGA comes in many sizes ranging from e.g. 10,000 to 8 Million system gates.
- Future FPGA generations should be easily utilizable.
  - Currently FPGA technology is evolving even faster than graphics processors!
FPGA prototyping boards:

- We have two suitable boards at IMM:

  - Celoxica RC-1000 reconfigurable computing platform:
    - 1 million gates, no multipliers, 16 kbytes on-chip RAM, 30 MHz.
    - 4x 32 bit SRAM memory (8 Mbytes)
    - PCI interface.

  - Avnet Xilinx Virtex-II development kit:
    - 4 million gates, 120 multipliers, 270 kbytes on-chip RAM, 100 MHz.
      - Space enough for up to 25 RISC CPU cores, 100 MHz each!
    - Up to 512 Mbytes 64-bit PC2100 DDR SDRAM module.
    - 64-bit 100 MHz PCI-X connector.
Case study:
The Hybris Graphics Architecture

- Developed at IMM, see my Ph.D. thesis.
- Hybrid-Parallel scalable graphics architecture.
- It is primarily a sort-middle architecture:
  - Tile-based architecture similar to GigaPixel, PowerVR, etc.
  - Screen is divided into 32x32 pixel tiles, triangles are sorted into bins for each tile.
  - Tiles can be rendered in parallel after sorting, which allows multiple tile rendering units to be used.
  - Framebuffer operations are fast (on-chip Z-buffer), but extra memory/bandwidth are needed for binning triangles.
- Several prototype implementations exist:
  - Parallel software rendering and FPGA-based graphics hardware.
Sort-middle parallelism with Tiles

Geometry database (random distribution)

G
G
G
G

Parallel Geometry Processors

Bucket sorted redistribution by screen region

R
R
R
R

Parallel Tile Renderers

Final image is assembled from subdivision tiles
Processes in the tile rendering engine

a) Setup Triangle

b) Draw Triangle

c) Setup Span

d) Draw Span

e) Final result
Pipelined tile rendering engine
FPGA floor plan

- Input controller
- Setup/Draw triangle
- VGA video processor
- Tile output controller
- Tile buffer 2
- Setup span
- Draw span
- Tile buffer 1
FPGA place & route floor plan
An Application Specific Reconfigurable Graphics Processor - Graphics Vision Day, IMM, DTU
Hardware rendered image example

- Image showing the Stanford “Dragon” laserscan rendered using the FPGA hardware implementation of the tile-based Hybris rendering architecture.
- The object contains 870k triangles and was rendered in 2 seconds on an experimental Xilinx Virtex 1000 FPGA board running at 25 MHz.
“Dragon” rendered by the FPGA
Next generation of our FPGA-GPU:

- Work-in-progress:
  - Design migration to next-generation FPGA architecture.
  - Change hardware description language from VHDL to SystemC.
  - PCI-X bus interface core on the FPGA.
  - DDR SDRAM memory controller.
  - 2x2 Interleaved pixel-parallel tile rendering engine, renders up to 4 pixels/clock.
  - Parallel tile rendering to render multiple tiles in parallel.
  - Programmable floating point vertex processor to run vertex shaders. Floating point units and MIPS-style RISC core implemented.

- A SystemC based simulator with a nice user interface allows easier debugging and cycle-true simulations.
2x2 pixel parallel tile rendering engine
Sparse super-sampling for anti-aliasing
Dual geometry / dual tile parallel implementation
Future?

- Future mass-market graphics processors may incorporate FPGA-like structures to allow:
  - Addition of new hardware features.
  - Fix hardware bugs “in the field”.
  - Flexible architecture, can be tuned to match the application.

- FPGA’s have revolutionized the way we think about hardware, by introducing “soft” hardware.
  - Like with software, there is also an opensource hardware movement. ([www.opencores.org](http://www.opencores.org))
  - The border between software and hardware has been blurred.