GPUs: Applications of Computer Arithmetic in 3D Graphics

RNC7

Stuart Oberman
Outline

- What is a GPU?
- 3D Graphics Pipeline Overview
- Arithmetic Formats and Representations in GPUs
- Fixed-Function Arithmetic Units
- Programmable Arithmetic Units
- Texture Units
- Future Research and Challenges
What is a GPU?
Generate Images > 60 FPS
Soul of the GPU

- Synthesize photorealistic images in real-time
  - > 60 frames per second
- Millions of pixels per frame can all be operated on in parallel
  - 3D graphics is often termed *embarrassingly parallel*
- Use large arrays of floating point units to exploit wide and deep parallelism
- Goal is to approach the image quality of movie studio offline rendering farms, but in real-time
  - Instead of hours per frame, > 60 frames per second
State-of-the-Art Film Graphics

- Offline rendering image quality in 2005, requiring hours / days per frame
- GPUs evolving to provide similar image quality at many frames per second
GPU Physical Comparison
GeForce 7900GTX, released in 2006

- 278 M transistors
- 650 MHz pipeline clock
- 196 mm² in 90nm
  - Intel Conroe 140mm² in 65nm
- >300 GFLOPS peak, single-precision
  - Intel Conroe with 128b SSE
  - 8 FLOPs/clk/core = 48 GFLOPs @ 3GHz
  - GPU > 6x FP throughput
Graphics Terminology

- **World space**
  - Initial orientation and arrangement of input geometry in 3D space

- **Eye space**
  - A 3D coordinate system based on the position and orientation of a virtual camera observing the geometry

- **Screen space**
  - The 2D representation of the scene after projection of the 3D scene into 2D space and conversion into screen format

- **Texel**
  - The smallest element of a textured 3D surface

- **Rasterization**
  - Converting a vertex representation to a pixel representation
Image Synthesis

Scene described by triangles of materials simulated by
- sampled images – textures
- numerically approximated properties

Vertex processing – independent vertex work
- screen position & attributes calculation
- example attributes: color, texture coordinates

Assemble and sample triangles
- generate pixels

Pixel processing – independent pixel work
- texture sampling, color calculation, visibility, and blending
The Life of a Triangle in a GPU

- process commands convert to FP
- transform vertices to screen-space
- generate per-triangle equations
- generate pixels, delete pixels that cannot be seen
- determine the colors, transparencies and depth of the pixel
- do final hidden surface test, blend and write out color and new depth

Host / Front End / Vertex Fetch

Vertex Processing

Primitive Assembly, Setup

Rasterize & Zcull

Pixel Shader

Texture

Pixel Engines (ROP)
A Tour of the NVIDIA 7900GTX GPU

- Host / FW / VTF
- Cull / Clip / Setup
- Shader Instruction Dispatch
- Fragment Crossbar
- Memory Partition
- DRAM(s)

- vertex fetch engine
- 8 vertex shaders
- conversion to pixels
- 24 pixel shaders
- redistribute pixels
- 16 pixel engines

4 independent 64-bit memory partitions
Numeric Representations in a GPU

- **Fixed point formats**
  - u8, s8, u16, s16, s3.8, s5.10, ...
  - Number of integer vs. fraction bits chosen based on particular unit

- **Floating point formats**
  - fp16, fp24, fp32, ...
  - Tradeoff of dynamic range vs. precision
  - New APIs require FP ops in programmable shaders to be IEEE compliant fp32

- **Block floating point formats**
  - Treat multiple operands as having a common exponent
  - Allows a tradeoff in dynamic range vs storage and computation
Choosing a Representation

For a given unit in a GPU, how to choose precision and representation?

Where API / IEEE Standard have requirements, choice is straightforward with no analysis required.

For other units, a typical algorithm is:
- Size and complexity minimization is primary goal
- Integer: is it good enough?
- Fixed-point with integer and fractional bits: is it good enough?
- Full floating-point?
- Iterative process going back-and-forth between fixed-point and floating-point
- Detailed analysis of images to guide decision process
Example: Motivation for fp16 in Texture Filtering, or High Dynamic Range (HDR)

- Light transport
  - Takes input geometry, texture maps, light positions, light radiances
  - Output is high dynamic-range per-pixel radiance value
  - Information stored in framebuffer with enough precision and range to represent wide range of intensity values
- 32b per-pixel was used previously as format for texture filtering, with four 8b integer values for red, green, blue, and alpha channel.
- Modern GPUs now use 64b per-pixel format
  - Each channel represented in fp16, or SM10e5 floating-point: sign+5b exponent+10b fraction
- Industry standard OpenEXR developed by Industrial Light & Magic: www.openexr.com
Benefits of fp16 for Light Transport

- 8b integer per channel
  - 8b ints provide only 100:1 difference in light source intensity; note blown-out look on windows and floor

- fp16 per channel
  - fp16 provides 9000:1 difference in light source intensity; note subtle lighting variations
Example: High Dynamic Range Scene

Image courtesy of Paul Debevec
Vertex Shaders

- Host / FW / VTF
- Cull / Clip / Setup
- Shader Instruction Dispatch
- Fragment Crossbar
- Memory Partition
  - DRAM(s)
- 8 vertex shaders

RNC7 July 10, 2006
Programmable Shaders

A shader is a small user-defined program that is executed within a GPU pipeline stage

- **Vertex Shader**: Shader executed in the vertex engine
- **Fragment or Pixel Shader**: Shader executed in the fragment engine

When active, a shader replaces fixed-function processing for its pipeline stage

The term is an anachronistic misnomer inherited from studio rendering software (RenderMan, etc.)

Shaders can do much more than just shading!
Vertex Shader Uses: Transform Vertex Positions

Why transform vertices?
- Rotate, translate and scale each object to place it correctly among the other objects that make up the scene *model*.
- Rotate, translate, and scale the entire scene to correctly place it relative to the camera’s position, view direction, and field of view.

How?
- Multiply every floating point vertex position by a combined 4x4 model-view matrix to get a 4-D \([x \ y \ z \ w]\) eye-space position.
Vertex Shader: Typical Lighting Vector Operations

Normalize to unit length

- Unit length vectors give useful results under dot products
- \( \text{length} = \sqrt{x^2 + y^2 + z^2} \)
- Divide each of \( x \), \( y \), and \( z \) by the length
  - preserves direction, length becomes 1.0
Fixed-Function Arithmetic: Clip, Cull, Triangle Setup, Rasterization

conversion to pixels
Fixed-Function Divide for Perspective

Why divide?
- Realistic perspective implies closer objects appear larger than faraway objects.
- In the graphics pipeline we perform a perspective divide on every vertex.
  - Divide all position components by the “w” term.
  - \([x \ y \ z \ w]\) becomes \([x/w \ y/w \ z/w \ 1]\).
  - All \(x/w\), \(y/w\), \(z/w\) after clipping and perspective divide are in the *normalized* range \([-1.0 , +1.0]\).
  - Typically implemented by reciprocation.
Rasterization

- Given a triangle, identify every pixel that belongs to that triangle

Point Sampling

- A pixel belongs to a triangle if and only if the center of the pixel is located in the interior of the triangle
- Evaluate 3 edge equations of the form $E = Ax + By + C$, where $E = 0$ is exactly on the line, and positive $E$ is towards the interior of the triangle.
- Design challenge is to implement these equations with sufficient precision, while minimizing latency, area, and design complexity
Plane Equation Solver

- Use plane equation to represent variation of attribute across triangle in 2D screen space
  \[ Ax + By + C = P \]
- Inputs sample points are attribute values at the three triangle vertices
- Solve for the three coefficients A, B, and C based on three samples
- Need fixed-function unit to compute fp32 coefficients A, B, C per attribute
Plane Equation Solver: Example of Usage of Compound Arithmetic Units

- Pipelined unit computes new set of fp32 A, B, C coefficients for one attribute / plane per clock
- Cross-product arithmetic requires 6 FP muls and 6 FP adds
- Optimize implementation for area and latency using fused and dot product operators
  - MAD: AxB + C
  - DP2: AxB + CxD,   DP3: AxB + CxD + ExF
  - FADD3: A + B + C
- Internal precision, rounding, and range of fused operators is flexible and application-specific
Pixel Shaders

24 pixel shaders
Pixel Shader Programmer’s View

From Previous Stage

Temporary Registers

Input Parameters

Texture Lookup & Filter

Add/Sub/Mul/Dot/Recip/...

Output Registers

To Next Stage
7900GTX Pixel Processor Detail

Shader Unit 1
4 FP MAD Ops / pixel
Dual/Co-Issue
Texture Address Calc
Free fp16 normalize
+ mini ALU

Texture Filter
Bi / Tri / Aniso
1 texture @ full speed
4-tap filter @ full speed
16:1 Aniso w/ Trilinear (128-tap)
FP16 Texture Filtering

Shader Unit 2
4 FP MAD Ops / pixel
Dual/Co-Issue
+ mini ALU
Shader Execution Units:
MAD Unit: Multiply-Add

- MAD unit operates on fp32 operands, produces fp32 output
- Performs all fundamental FP operations:
  - FADD, FMUL, FMAD
- Fully-pipelined, but latency is not over-optimized
- FADD and FMUL implemented close to IEEE standard
  - Denorms are flushed-to-zero
  - Special numbers properly handled
- FMAD different from FMA
  - Intermediate product is kept to less than full width
  - For graphics, this is sufficient precision, and it provides a 2x increase in FP throughput for lower cost
Shader Execution Units: Attribute Interpolator

- Plane equation unit generates plane equation fp32 coefficients to summarize all triangle attributes.
- A, B, and C are fp32 interpolation parameters associated with a given triangle’s attribute U.
- Resulting attribute value U is fp32.
- Pixel shader hardware must interpolate value of each attribute per (x,y) for all pixels to be drawn: \( U(x,y) = A \times x + B \times y + C \).
- For perspective correct interpolation:
  - Interpolate 1/w, and reciprocate to form w.
  - Interpolate U/w.
  - Multiply U/w and w form perspective-correct U.
Shader Execution Units:
Special Functions Unit

- Shader hardware designed to support OpenGL and DirectX APIs, including several high-order functions
- APIs require at least
  - rcp, rsqrt
  - log2, exp2
  - sin, cos
- High parallelism implies desire for high throughput
  - Desire for function evaluator to be fully pipelined
- Near fp32 accuracy required
  - rcp to 1ulp, rsqrt to 2ulps
Quadratic Interpolation Algorithm

Based on Enhanced Minimax Approximation (Pineiro and Oberman 2005)

For n-bit input $X$, approximate

$$f(X) \approx C_0 + C_1 X_l + C_2 X_l^2$$

Divide $X$ into $m$-bits $X_u$ and $n-m$ bits $X_l$

Upper $m$-bits index into three tables to return three coefficients $C_0$, $C_1$, and $C_2$

Use three step hybrid coefficient generation process, based on minimax approximation, that accounts for all approximation and truncation errors
### Special Function Statistics in Modern GPUs (ARITH17)

<table>
<thead>
<tr>
<th>Function</th>
<th>Input Interval</th>
<th>M</th>
<th>Configuration</th>
<th>Accuracy (good bits)</th>
<th>Ulp error</th>
<th>% exactly rounded</th>
<th>Monotonic</th>
<th>Lookup table size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/X</td>
<td>[1,2)</td>
<td>7</td>
<td>26,16,10</td>
<td>24.02</td>
<td>0.98</td>
<td>87%</td>
<td>Yes</td>
<td>6.50Kb</td>
</tr>
<tr>
<td>1/sqrt(X)</td>
<td>[1,4)</td>
<td>6</td>
<td>26,16,10</td>
<td>23.40</td>
<td>1.52</td>
<td>78%</td>
<td>Yes</td>
<td>6.50Kb</td>
</tr>
<tr>
<td>$2^x$</td>
<td>[0,1)</td>
<td>6</td>
<td>26,16,10</td>
<td>22.51</td>
<td>1.41</td>
<td>74%</td>
<td>Yes</td>
<td>3.25Kb</td>
</tr>
<tr>
<td>log$_2$X</td>
<td>[1,2)</td>
<td>6</td>
<td>26,16,10</td>
<td>22.57</td>
<td>n/a</td>
<td>n/a</td>
<td>Yes</td>
<td>3.25Kb</td>
</tr>
<tr>
<td>Sin/cos</td>
<td>[0,pi/2)</td>
<td>6</td>
<td>26,15,11</td>
<td>22.47</td>
<td>n/a</td>
<td>n/a</td>
<td>No</td>
<td>3.25Kb</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>22.75Kb</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Desire fully-pipelined performance
- Use quadratic interpolation for fast and accurate estimates to the special functions
Multifunction Interpolator (ARITH17)
Texture Unit: A GPU’s Load Unit

- Host / FW / VTF
- Cull / Clip / Setup
- Shader Instruction Dispatch
- Z-Cull
- Fragment Crossbar
- Memory Partition
  - DRAM(s)
- 6 texture units
- 2 Tex

6 texture units
Texture Mapping

- Associate points in an image to points in a geometric object
- Blend texture color data with interpolated color
What is a Texture?

- Index into a 2D array with shader interpolated floating point index: parameterized surface
- Integer or FP filtering performed on the returned samples / texels
- Bilinear filtering is most common method:

```
   t0     t1
   
   t2
```

- Blending arithmetic:
  \[
  \text{tex}(x, y) = (1-t) \left( (1-s) t0 + s t1 \right) + t \left( (1-s) t2 + s t3 \right)
  \]
Texture: Mip Maps

- Maintain texture at coarse levels of detail – each half the size of the one before
- Sample between level pairs, weighting according to fractional level of detail
  - Trilinear filtering
- Problem:
  - Blurry when footprint is not square
Texture Pipeline Operation

Pipelined texture functional unit must complete these steps at high speed, fully-pipelined:

1. Receive texture address \((s, t)\) for the current screen pixel \((x, y)\), where \(s\) and \(t\) are represented as fp32.
2. Calculate the texture minification, \(j\)
3. Extract level of detail or MIP-map levels to be used
4. Calculate trilinear interpolation fraction, \(f\)
5. Scale texture address \((s, t)\) for the levels selected
6. Access memory and retrieve desired texels
7. Perform appropriate filtering operation on texels and return results
Pixel to Texel Mapping

Fig. 1. Pixel inverse mapped to texture-space.

Fig. 2. Pixel inverse mapped to texture-space using constant partial derivatives.
Computing Level of Detail: One Method

\[ j = \max(|r_1|, |r_2|) \]

where

\[ |r_1| = \sqrt{s_x^2 + t_x^2} \]

and

\[ |r_2| = \sqrt{s_y^2 + t_y^2} \]

\[ \therefore j = \sqrt{\max(s_x^2 + t_x^2, s_y^2 + t_y^2)} \]

where \( j \) = texel : pixel minification ratio.

- \( s_x, s_y, t_x, t_y \) and partial derivatives computed on FP32 numbers

- Several methods for implementation
Texture Filtering: Bilinear

Nearest

Bilinear
Texture Filtering: Anisotropic Sampling

- If filter footprint is not square, take multiple samples over footprint pattern
- More complicated arithmetic
- Step through samples
- Weight, blend and accumulate arithmetic pipeline
- Higher image quality
- Lower performance
Texture Filtering: Anisotropic
Conclusions

- GPUs contain significant arithmetic computation to exploit extreme parallelism
- Wide variety of arithmetic functional units and representations in a GPU
  - Shaders and fixed-function units
  - Trend towards programmability
- Ever-increasing performance and features
  - GPUs endeavor to provide photorealistic imagery in real-time
  - Today’s GPUs provide more than 300 GFLOPs of single precision and are increasing rapidly
  - Newest CPU by comparison provides 50 GFLOPs
Opportunities for Future Research

- Total graphics performance is often a weak function of arithmetic unit latency.
  - Opportunities for optimization for area and power?
- Performance / watt is key metric for future designs.
  - We are at power supply and thermal limits
  - Arithmetic algorithms and implementations optimized for perf/watt?
  - Best numeric representations for perf/watt?
- Closed-form analysis of arithmetic precision requirements for various intermediate stages
  - Rasterization, clipping, plane equation generation, interpolation
Opportunities for Future Research

- Higher precision arithmetic
  - fp64 and fp128 in GPUs? Example usage models include large scene databases
  - Efficient implementations when shared with narrower datatypes?
- Compression: maximize effective memory bandwidth and footprint
  - Texture maps: lossy Microsoft’s DXT, and others
  - Lossless color compression in framebuffer
  - Non-linear fixed-point representations
    - Example: sRGB, more visually uniform
  - Lossy and lossless compression of floating point
Opportunities for Future Research

GPGPU

General Purpose Computing on GPU, is a new field focusing on use of arithmetic power in GPUs for general scientific computation.

Field has gained significant attention by scientific researchers looking to harness the ever-increasing GPU computing power for non-graphics applications.