Speed-up of Algorithms With Graphics Processing Units (GPU): Part I of IV

*# Derek Anderson and *# Robert Luke

* Electrical and Computer Engineering Department
# Predoctoral Fellows, NLM Training Grant

IEEE Computational Intelligence Society MU Chapter
And
National Library of Medicine Medical Informatics Training Grant
Special Seminar Series
Organization of Lectures

- **Part I**
  - Introduction to GPUs and shader languages
- **Part II**
  - Image processing (Morphology, Sobel, and Gaussian)
- **Part III**
  - Performance, multi-pass rendering, optimizations, and debugging
- **Part IV**
  - Using GPUs for non-image based processing (SOFM & CA)
Motivation: Why GPUs?

- Traditionally, most graphics operations, such as mathematical transformations between coordinate spaces, rasterization, and shading operations have been performed on the CPU.
- There is a need to offload many of these operations from the CPU (primarily arithmetic and logic) to specialized graphics hardware (based on vector & matrix processing).
Motivation: Why GPUs?

- As graphics continue to advance, it is important that we have a greater degree of control over stages in the graphics pipeline
  - Don’t want fixed functionality anymore!
- Instead of designing hardware for each graphics algorithm (yea right!), GPUs were invented in order to generalize the pipeline and our interface to it
- Need for programmability and scalability
What is the Graphics Pipeline?
(very high level perspective)

Application – Driver Program

OpenGL  GLEW  GLUT/GLFW

CPU

CPU-GPU Interface/Bus (AGP, PCI Express)

GPU Front End  Primitive Assembly  Rasterization & Interpolation  Raster Operations  Frame Buffer

Programmable Vertex Processor  Programmable Fragment Processor

GPU
Visualizing the Pipeline
Example: Per-Pixel Shading
Important Concepts

- Pipelining
  - Number of stages

- Parallelism
  - Number of parallel processes

- Parallelism + pipelining
  - Number of parallel pipelines
7800 Architecture
History of GPUs

- **Application Domains**
  - Gaming
  - Film
  - Computer Aided Design

- **Pre-GPU Graphics Acceleration**
  - Companies (such as SGI) offered specialized & expensive graphics hardware
  - Did not achieve mass-market success

- **First-Generation GPUs (up to 1998)**
  - NVIDIA’s TNT, ATIs Rage, and 3dfx’s Voodoo3
  - Capable of rasterizing pre-transformed triangles and applying one or two textures
  - Lacked ability to transform vertices
  - Limited set of math operations for combining textures to compute the color of rasterized pixels

The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics
History of GPUs

  - NVIDIA's GeForce 256 & GeForce2, ATI’s Radeon 7500, and S3’s Savage3D
  - Offloaded vertex transformations and lighting from the CPU to the GPU
  - Was configurable, but still not truly programmable

- Third-Generation GPUs (2001)
  - NVIDIA’s GeForce3 and GeForce4 Ti, Microsoft’s Xbox, and ATI’s Radeon 8500
  - Vertex programmability finally!
  - Pixel level configurability, but not truly configurable

- Fourth-Generation GPUs (2002 to present)
  - NVIDIA’s GeForce FX family (6800, 7800, …), ATI’s Radeon 9700 …
  - Provide vertex and fragment programmability
  - Newer NVIDIA 7950’s and ATI Radeon® X1950

- Next Generation GPUs?
  - Unified Shader Architecture (vertex, geometry, and fragment processors)
NVIDIA and ATI (AMD Acquired!)

- NVIDIA 7800 ($300-$500)
- Radeon X1950 ($500)
- NVIDIA Quadro FX 5500 ($2,500)
- NVIDIA Quadro Plex ($17,500?)
Growth & Development

- Moore’s Law
  - Empirical observation that the rate of technological development, the complexity of an integrated circuit, with respect to minimum component cost, will double every 18 months

- GPUs are getting faster
  - CPUs ≈ 1.4 annual growth
  - GPUs ≈ 1.7 (pixels) to 2.3 (vertices) annual growth

- Measuring the number of GFLOPs
  - 3.0 GHz dual-core Pentium4
    - 24.6 GFLOPS
  - NVIDIA GeForceFX 7800
    - 165 GFLOPs
ATI Xenos - XBOX 360 GPU

- 337 million transistors
- 500 MHz parent GPU
- Max poly performance: 500 million triangles per second
- 16 filtered or unfiltered texture samples per clock
- 48-way parallel floating point dynamically-scheduled shader pipelines
  - Unified shader architecture
  - 160 programmable shader operations per cycle (48 ALUs x 2 ops + 16 texture fetches + 32 control flow + 16 vertex fetch)
  - 48 billion shader operations per second
  - 240 GFLOPS
Shader Languages

• How do you program for a GPU?
• Cg (NVIDIA)
  – C for Graphics
• GLSL (OpenGL)
  – OpenGL Shading Language
• HLSL (Microsoft)
  – High Level Shading Language
• Which one do you pick?
• Pick one or support all?
  – Console (fixed hardware)
  – PC (hardware varies greatly!)
OpenGL Shader Language

• Also known as GLslang
• It was created by the OpenGL ARB to give developers more direct control of the graphics pipeline without having to use assembly language or hardware-specific languages (such as NVIDIA's Cg shader language)
• GLSL is a high-level procedural language
• Has its roots in C
• Stronger type checking than C
• Same language, with subtle differences, is used for both vertex and fragment shaders
Cg Language

- Cg is NVIDIA's open-source high-level shading language
- Cg was developed in close collaboration with Microsoft and is syntactically equivalent to HLSL (the shading language in DirectX 9)
- Cg replaces assembly code with a C-like language and a compiler

Cg: http://developer.nvidia.com/page/cg_main.html
Compiling and Loading Shaders

Cg Program Text → Cg Compiler
Cg Runtime API → CgGL Runtime API
GPU Assembly → OpenGL Driver

Application → Shader source code
OpenGL API

Compiler → Shader Object
Linker → Program Object

Compiled code → Executable code

Graphics Hardware

GLSL
General Purpose GPU (GPGPU) Programming

Speed up of non-image based applications
- General Computation
- Linear Algebra
- Differential Equations

Image Processing & Computer Vision

Pattern Recognition
- Clustering (we are the ones pushing this!)
- SOFM

Sorting & Searching Algorithms
What Do GPUs Support?

- Texture Sampling & Transformation
- Vector & Matrix Operations
  - length (Euclidean length of a vector)
  - distance (Euclidean distance between two points)
  - normalize (vector norm)
  - vector and matrix multiplication
  - dot product
  - cross product
  - transpose
- Trig Functions
- Power & Log (2, 10, natural) Functions
- Lerp (Linear Interpolation)
- Misc. Operations (max, min, any, all, …)
Cg Program: Per-Fragment Shading

Vertex Program

void v_lighting( float4 position : POSITION,
                float3 normal : NORMAL,
                out float4 oPosition : POSITION,
                out float3 objectPos : TEXCOORD0,
                out float3 oNormal : TEXCOORD1,
                uniform float4x4 modelViewProj )
{
    oPosition = mul(modelViewProj, position);
    objectPos = position.xyz;
    oNormal = normal;
}
Cg Program: Per-Fragment Shading

**Fragment Program**

```c
void f_lighting( float4 position : TEXCOORD0 ,
                 float3 normal : TEXCOORD1 ,
                 out float4 color : COLOR ,
                 uniform float3 lightPosition ,
                 uniform float3 eyePosition ,
                 uniform float shininess )
{
    float3 P = position.xyz;
    float3 N = normalize(normal);
    float3 ambient = float3( 1.0 , 0.5 , 0.0 );
    float3 L = normalize( lightPosition – P );
    float3 diffuseLight = max( dot( L , N ) , 0.0 ) * float3(1.0,1.0,1.0);
    float3 V = normalize(eyePosition – P);
    float3 H = normalize(L+V);
    float3 spec = pow(max(dot(N,H),0.0),shininess) * float3(1.0,1.0,1.0);
    color.xyz = ambient + diffuseLight + spec;
}
```