Using the GPU In Game Programming

Quick aside...

► Joseph T. Kider Jr.

► Lecturer
  ► GPU Programming
  ► Physically Based Animation
  ► Senior Design

► Associate Director
  ► Manage lab projects

► PhD Candidate
  ► Decay and Destruction
  ► Multimodal Motion Capture

► For more information please visit: http://www.seas.upenn.edu/~kiderj

Quick aside....

► Computer Graphics @ Penn

► Officially founded in 1990
  ► Though Dr. Badler has been there since 1975 working on Character animation
  ► 10 PhD students, 3 faculty, 4 visiting scholars

► Digital Media Design Program
  ► 80 engineering undergraduates

► Computer Graphics and Gaming Masters
  ► 45 master students

► For more information please visit: http://cg.cis.upenn.edu/
Penn’s Masters Program…

► Computer Graphics and Game Technology
  ► Master of Science in Engineering
  ► 10 courses: Computer Science, Business, Art, …

► Core Areas of Study
  ► 1) Computer Science, Systems and Technology
    ► covers mathematics, physics, artificial intelligence, programming, algorithm design, computer graphics and animation technology, and game design and development.
    ► C / C++ / C# / Python

  ► 2) Creative Arts and Design
    ► This core area is intended to address issues related to designing, and creating visual effects.
    ► Maya, Zbrush …

Our Graduates…

► Pixar
  ► Samantha Raja, Daniel Garcia, …

► Dreamworks
  ► Terry Kalelas, Jeremy Cytryn, Varun Talwar, …

► Electronic Arts
  ► Steven Lovejoy, Bobby Wilkinson, …

► Zynga, Facebook, Twitter

► Rockstar

► Sony

► Microsoft (Games) Studio

► Blue Sky
Example of Our Classes…

CIS563: Physically Based Animation

- Explores physically based simulation methods for computer animation of a wide variety of phenomena and materials
  - Rigid and deformable solids, cloth, liquids, virtual characters, and explosions.
  - Numerical methods, physical models, data structures, vector calculus.

- http://www.seas.upenn.edu/~cis563/

Instructor: Joseph T. Kider
TAs: Aline Normoyle (Disney Research), Varun Talwar (Dreamworks), Terry Kalses (Dreamworks)
Our classes... (Student Work: Dan Garcia - Pixar)

Our classes... (Student Work: Raul Santos)

Back to our regular scheduled broadcasting....

Guest Lecture: Joseph Kider (CS 15-466: Game Programming in Fall 2011)
Hands on exercise....

- Parallel Sorting
  - Get into groups
  - 10 minutes

Call of Duty: MMF

http://badcompany2.ea.com/

Guest Lecture: Joseph Kider (CS 15-466: Game Programming in Fall 2011)
Overview

- Motivation
- Introduction to the GPU
- Evolution of the Graphics Pipeline
- Introduction to Shaders
- Introduction to CUDA
- What we can do with the GPU in Games
- Summary...

Tasks

- Games compute many ‘tasks’ at once
  - AI
  - Path Planning
  - Skinning
  - Rendering / Lighting
  - Explosions, fire, smoke
  - Rigid body dynamics
  - ....
Sources.... (more detailed at the end)

- David Kirk (NVIDIA)
- Wen-mei Hwu (UIUC)
- David Lubke, Johan Andersson
- Wolfgang Engel, Nate Carr
- Suresh Venkentansuremian (Utah)
- Programming Massively Parallel Processors
- The CG Tutorial
- GPU Gems 1, 2, 3
- Patrick Cozzi, Jon McCaffrey, Gary Katz

Design Goals++

- System should allow for fine-grained control
  - Without being pedantic and labor-intensive
- Decay Phenomena should act with some physical-biological-chemical basis
  - But should also allow for "artistic direction"
- We want accessible control structures
  - Able to age and decay a variety of levels of detail

What is a GPU?

GPU: Graphics Processing Unit
Processor that resides on your graphics card.

GPUs allow us to achieve the unprecedented graphics capabilities now available in games

What is Parallel Computing?

- Parallel computing: using multiple processors to...
  - More quickly perform a computation, or
  - Perform a larger computation in the same time
  - PROGRAMMER expresses parallelism

Clusters of Computers: MPI, networks, cloud computing.... NOT COVERED

Shared memory Multiprocessor
Called "multicore" when on the same chip
Xbox, Cell, Playstation 3,... NOT COVERED

GPU: Graphics processing units FOCUS OF TODAY!

Slide courtesy of Milo Martin
What is Parallel Computing?

- Parallel Computing has been around a long time!!
- 1972 Cray Supercomputers
- 1992 VLSI processor
- 1999 Pentium 3 SSE Extensions

- So why is Parallel programming so popular now?

Was is GPU Programming in?

- Games demand advanced shading
- Fast GPUs = better shading
- Need for speed = continued innovation
- The gaming industry has overtaken the defense, finance, oil and healthcare industries as the main driving factor for high performance processors.

Where is the money?

- Medical Science Visualization Healthcare
- LIFE

MONEY
Where is the money?

LIFE

DARPA
Lockheed Martin

DEATH

Where is the money?

LIFE

Wii sports
$14,278,054

Battlefield 3
$2,320,118

DEATH

VIDE O GAMES !?!

Batman
$1,592,586

Why Program the GPU?

- Compute
- Memory Bandwidth
- Install Base
Why Program the GPU?

- **Compute**
  - Intel Core i7 – 4 cores – 100 GFLOP
  - NVIDIA GTX280 – 240 cores – 1 TFLOP
  - NVIDIA GTX280 – 240 cores – 1 TFLOP

- **Memory Bandwidth**

- **Install Base**

---

Why Program the GPU?

- **Compute**
  - Intel Core i7 – 4 cores – 100 GFLOP
  - NVIDIA GTX280 – 240 cores – 1 TFLOP
  - NVIDIA GTX280 – 240 cores – 1 TFLOP

- **Memory Bandwidth**
  - System Memory – 60 GB/s
  - NVIDIA GT200 – 150 GB/s

- **Install Base**

---

GPU = Fast co-processor

- GPU speed increasing at cubed-Moore’s Law.
- This is a consequence of the *data-parallel streaming* aspects of the GPU.
- GPUs are cheap! Put a couple together, and you can get a super-computer.

So can we use the GPU for general-purpose computing and game programming?

NYT May 26, 2003: TECHNOLOGY; From PlayStation to Supercomputer for $50,000:
National Center for Supercomputing Applications at University of Illinois at Urbana-Champaign builds supercomputer using 70 Individual Sony Playstation 2 machines; project required no hardware engineering other than mounting Playstations in a rack and connecting them with high-speed network switch.
Yes ! Wealth of applications

- Data Analysis
- Motion Planning
- Voronoi Diagrams
- Geometric Optimization
- Particle Systems
- Force-field simulation
- Molecular Dynamics
- Graph Drawing
- Physical Simulation
- Matrix Multiplication
- Conjugate Gradient
- Database queries
- Sorting and Searching
- Range queries
- Image Processing
- Signal Processing
- Radar, Sonar, Oil Exploration
- Finance
- Optimization
- Planning
- ... and graphics too !!

Overview

- Motivation
- Introduction to the GPU
- Evolution of the Graphics Pipeline
- Introduction to Shaders
- Introduction to CUDA
- What we can do with the GPU in Games
- Summary…

GPU Pipeline

Guest Lecture: Joseph Kider (CS 15-466: Game Programming in Fall 2011)
GPU Pipeline

- VERTICES
- Rasterized
- Colored Fragments

Stream Programming

- A **stream** is a sequence of data (could be numbers, colors, RGBA vectors, etc.)
- A **kernel/shader** is a (fragment, vertex, geometry) program that runs on each element of a stream, generating an output stream (pixel buffer).

- **Kernel** = vertex/fragment/… shader
- **Input stream** = stream of vertices, primitives, or fragments
- **Output stream** = frame buffer or other buffer (transform feedback)
- **Multiple kernels** = multi-pass rendering sequence on the GPU.
Stream Programming

To program the GPU, one must think of it as a (parallel) stream processor.

What is the cost of a stream program?

- Number of kernels
  - Readbacks from the GPU to main memory are expensive, and so is transferring data to the GPU.
- Complexity of kernel
  - More complexity takes longer to move data through a rendering pipeline
- Number of memory accesses
  - Non-local memory access is expensive
- Number of branches
  - Divergent branches are expensive

GPU Evolution...

Generation I: 3dfx Voodoo (1996)
- One of the first true 3D game cards
- Worked by supplementing standard 2D video card.
- Did not do vertex transformations: these were done in the CPU
- Did do texture mapping, z-buffering.

Generation II: GeForce/Radeon 7500 (1998)
- Main innovation: shifting the transformation and lighting calculations to the GPU
- Allowed multi-texturing: giving bump maps, light maps, and others...
- Faster AGP bus instead of PCI
Guest Lecture: Joseph Kider (CS 15-466: Game Programming in Fall 2011)
GPU Evolution...

Why Unify Shader Processors?

Unified Shader Processors

why unify?

NVIDIA G80 Architecture

GeForce 8: Modern GPU Architecture

Slide from David Luebke: http://s08.idav.ucdavis.edu/luebke-nvidia-gpu-architecture.pdf

GPU Evolution...

why unify?

NVIDIA G80 Architecture

GeForce 8: Modern GPU Architecture

Slide from David Luebke: http://s08.idav.ucdavis.edu/luebke-nvidia-gpu-architecture.pdf

GPU Evolution...

Unified Shader Processors

Dynamic Load Balancing – Company of Heroes

Less Geometry

More Geometry

Unified Shader

Slide from David Luebke: http://s08.idav.ucdavis.edu/luebke-nvidia-gpu-architecture.pdf

GPU Evolution...

Why Unify Shader Processors?

why unify?

Heavy Geometry Workload Part = 4

NVIDIA G80 Architecture

GeForce 8: Modern GPU Architecture

Slide from David Luebke: http://s08.idav.ucdavis.edu/luebke-nvidia-gpu-architecture.pdf

11/16/2011

Guest Lecture: Joseph Kider (CS 15-466: Game Programming in Fall 2011)
Deep Pipelines

- think assembly line conveyer belt

Parallelism, Parallelism, Parallelism

- Deep/Wide pipelines allow memory latency to be hidden.
  - If one fragment(or vertex) is waiting on a memory fetch, go work on another element for a while.
- Do not need expensive huge caches (unlike CPU’s)
  - Fill precious chip space with arithmetic logic units (ALU).
Current parallelism

Game development
- 2 year development cycle
  - New IP often takes much longer, 3-5 years
  - Engine is continuously in development & used
- AAA teams of 70-90 people
  - 50% artists
  - 30% designers
  - 20% programmers
  - 10% audio
- Budgets $20-40 million
- Cross-platform development is market reality
  - Xbox 360 and PlayStation 3
  - PC DX10 and DX11 (and sometimes Mac)
  - Current consoles will stay with us for many more years

Game Engine Requirements...

Levels of code in Frostbite

- Editor (C#)
- Pipeline (C++)
- Game code (C++)
- System CPU-jobs (C++)
- System SPU-jobs (C++/asm)
- Generated shaders (GLSL)
- Compute kernels (CUDA)

GLSL Syntax...
- GLSL is like C without
  - Pointers
  - Recursion
  - Dynamic memory allocation
- GLSL is like C with
  - Built-in vector, matrix and sampler types
  - Constructors
  - A great math library
  - Input and output qualifiers

Allow us to write concise, efficient shaders.
• GLSL has a preprocessor

```c
#define FAST_EXACT_METHOD

#ifdef FAST_EXACT_METHOD
    FastExact();
#else
    SlowApproximate();
#endif

// ... many others
```

• All Shaders have main()

```c
void main(void)
{
}
```

• Scalar types: float, int, uint, bool

• Vectors are also built-in types:
  - vec2, vec3, vec4
  - ivec*, uvec*, bvec*

• Access components three ways:
  - .x, .y, .z, .w  position or direction
  - .r, .g, .b, .a  color
  - .s, .t, .p, .q  texture coordinate

```c
myColor.xyz
myColor.xgb
```
Guest Lecture: Joseph Kider (CS 15-466: Game Programming in Fall 2011)

### GLSL Syntax (Vertex Shader)

**• One Vertex In, One Vertex Out**

- 3D Position
- Normal
- Tex Coords
- Color

**Vertex Shader**

- Light, Xform, Project

**3D Position Normal Tex Coords Color**

- Light, Xform, Project

**GLSL Syntax (Vertex Shader)**

```glsl
void phongShadeVertexProgram(
    float4 iCol : COLOR,
    float4 iPos : POSITION,
    float4 iNorm : NORMAL,
    float2 iTex0 : TEXCOORD0,
    out float4 oColor : COLOR,
    out float4 oScreenPos : POSITION
)
{
    oScreenPos = mul(glstate.matrix.mvp,iPos);
    oPos = mul(glstate.matrix.modelview[0],iPos).xyz;
    oN = normalize(mul(glstate.matrix.invtrans.modelview[0],iNorm).xyz);
    oTex = iTex0;oColor = iCol;
    return;
}
```

### GLSL Syntax (Fragment Shader)

**• One Fragment in (0-1 Fragments out)**

- 3D Position Normal Tex Coords Color Screen Pos, Depth
- Color, Depth
- Texture and Light

**Fragment Shader**

- Texture Maps

**GLSL Syntax (Fragment Shader)**

```glsl
void phongShadeFragmentProgram(
    float2 tex0 : TEXCOORD0,
    float3 P : TEXCOORD1,
    float3 N : TEXCOORD2,
    uniform sampler2D textureMap,
    uniform float3 Ka,Ks,Kd,LP,
    uniform float  Kp,
    out float4 oColor : COLOR
)
{
    N = normalize( N ); // Compute the diffuse term
    float3 L = normalize(LP - P);
    float diffuseLight = max(dot(L, N), 0);
    float4 Td = tex2D(textureMap,tex0.xy);
    float3 diffuse = float3(1.0,1.0,1.0) * Kd * diffuseLight;
    // Compute the specular term
    float3 V = normalize( -P);
    float3 H = normalize(L + V);
    float specularLight = pow(max(dot(H, N), 0), Kp);
    if (diffuseLight <= 0)
        specularLight = 0;
    float3 specular = Ks.xyz * specularLight;
    oColor.xyz = ( Ka*float3(1.0,1.0,1.0) + diffuse + specular);
    oColor.w = 1;
    return;
}
```

---

Slide curiosity of Nate Carr
GLSL Syntax (Multi-Pass)

Composite Effects

- Vertex Shader
- Triangle Setup
- Fragment Shader
- Frame Blender
- Frame-Buffer(s)

Transform
Project

Texture Maps

Combine vertices into triangle, convert to fragments

Texture map fragments

Z-cull
Alpha Blend

Frame-Buffer(s)

Render To Texture

Texture Maps

Render To Texture

Render To Vertex Array

GLSL Syntax (Environment Map)

GLSL Syntax (Environment Map)

Generated shaders [1]
- Graph-based surface shaders
- Treated as content, not code
- Artist created
- Generates HLSL code
- Used by all meshes and 3d surfaces

Graphics / Compute kernels
- Hand-coded & optimized GLSL / HLSL / CG
- Statically linked in with C++
- Pixel- & compute-shaders
- Lighting, post-processing & special effects

Shader Types...

Graph-based surface shader in FrostEd 2

Student Work CIS565 GPU Programming – UPENN – Joseph Kider

Robin Feng (CGGT 2011) 

Frank Chao (CGGT 2011)

Guest Lecture: Joseph Kider (CS 15-466: Game Programming in Fall 2011)
Overview

- Motivation
- Introduction to the GPU
- Evolution of the Graphics Pipeline
- Introduction to Shaders
- Introduction to CUDA
- What we can do with the GPU in Games
- Summary…

General Purpose Computing (GPGPU)

- Certain computations map poorly into shaders
- For example
  - Add an array

General Purpose Computing (GPGPU)

- What’s ‘general purpose’ computations does the GPU calculate?
  - Everything you can get it to do efficiently for you!!!
- Physics
- Collision Detection
- AI
- And yes, graphics too…

G80, GT200, GTX580 … (FERMI)

- November 2006:  G80
- June 2008:      GT200
- March 2011:    Fermi (GF100)
**FERMI Hardware**

GF100
- 16 SMs
- Each with 32 cores
  - 512 total cores
- Each SM hosts up to
  - 48 warps, or
  - 1,536 threads
- In flight, up to
  - 24,576 threads

**FERMI Architecture**

**Transistor Usage**


**“CPU-style” cores**

**Slimming down**


**Slide from:** http://bps10.idav.ucdavis.edu/talks/03-fatahalian_gpuArch/Teraflop_BPS_SIGGRAPH2010.pdf

---

**Guest Lecture:** Joseph Kider (CS 15-466: Game Programming in Fall 2011)
FERMI Architecture

Sixteen cores (sixteen fragments in parallel)

16 cores = 16 simultaneous instruction streams


FERMI Architecture

Add ALUs

Idea #2: Amortize cost/complexity of managing an instruction stream across many ALUs

SIMD processing


FERMI Architecture

128 fragments in parallel

16 cores = 128 ALUs, 16 simultaneous instruction streams


FERMI Architecture

Throughput!

Time (clocks)


Guest Lecture: Joseph Kider (CS 15-466: Game Programming in Fall 2011)
**FERMI Architecture**

### Storing contexts

- *Pool of context storage*
  - 128 KB

---

### Eighteen small contexts

- (maximal latency hiding)

---

### Twelve medium contexts

- (low latency hiding ability)

---

**Guest Lecture: Joseph Kider (CS 15-466: Game Programming in Fall 2011)**
FERMI Architecture

My chip!

16 cores
8 mul-add ALUs per core (128 total)
16 simultaneous instruction streams
64 concurrent (but interleaved) instruction streams
512 concurrent fragments
= 256 GFLOPs (@ 1GHz)


GPU Programming Model

• A kernel is executed as a grid of thread blocks
  – All threads share data memory space
• A thread block is a batch of threads that can cooperate with each other by:
  – Synchronizing their execution
    • For hazard-free shared memory accesses
  – Efficiently sharing data through a low latency shared memory
• Two threads from two different blocks cannot cooperate

Overview

► Motivation
► Introduction to the GPU
► Evolution of the Graphics Pipeline
► Introduction to Shaders
► Introduction to CUDA
► What we can do with the GPU in Games
► Summary…

Diving In

#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/sort.h>

int main(void)
{
  // generate 1M random numbers on the host
  thrust::host_vector<int> h_vec(1 << 24);
  thrust::generate(h_vec.begin(), h_vec.end(), rand);

  // transfer data to the device
  thrust::device_vector<int> d_vec = h_vec;

  // sort data on the device
  thrust::sort(d_vec.begin(), d_vec.end());

  // transfer data back to host
  thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

  return 0;
}
Audience Participation Game

Calculate the Point Directions:
Here is an example of how to calculate the local direction of edges.
Overview

► Motivation
► Introduction to the GPU
► Evolution of the Graphics Pipeline
► Introduction to Shaders
► Introduction to CUDA
► What we can do with the GPU in Games
► Summary…

Future Hardware….

• 2015 = 50 TFLOPS, we would spend it on:
  – 80% graphics
  – 15% simulation
  – 4% misc
  – 1% game (wouldn’t use all 500 GFLOPS for game logic & glue!)

• OOE CPUs more efficient for the majority of our game code
  – But for the vast majority of our FLOPS these are fully irrelevant
  – Can evolve to a small dot on a sea of DP cores
  – Or run on scalar ISA wasting vector instructions on a few cores

• In other words: no need for separate CPU and GPU!

Conclusions….

• Developer productivity can’t be limited by model
  – It should enhance productivity & perf on all levels
  – Tools & language constructs play a critical role
  – Lots of opportunity for innovation and standardization!

• We are willing to go great lengths to utilize any HW
  – If that platform is part of our core business target and can makes a difference

• We for one welcome our parallel future!
More Resources....

- CIS565: GPU Programming - UPENN
  - [http://www.seas.upenn.edu/~cis565/](http://www.seas.upenn.edu/~cis565/)
- CMU 15-869: Graphics and Imaging Architectures – CMU
- Applied Parallel Programming – UIUC
  - [http://courses.engr.illinois.edu/ece498/sp/](http://courses.engr.illinois.edu/ece498/sp/)
- GPU Programming – USCD
  - [http://graphics.ucsd.edu/twiki/bin/view.pl/Classes/CSE190Winter2010](http://graphics.ucsd.edu/twiki/bin/view.pl/Classes/CSE190Winter2010)
- Beyond Programmable Shading – Stanford
  - Look for podcasts on itunes

More Resources....

  - GPU Gems 1,2,3 ; CG Tutorial
- Not free but good books
  - Programming Massively Parallel Processors
  - CUDA by Example
  - ShaderX Series (1,2,3,4,5,6...)
  - GPU Computing Gems
  - 3D Engine Design for Virtual Globes

Questions?

[http://www.seas.upenn.edu/~kiderj](http://www.seas.upenn.edu/~kiderj)

Guest Lecture: Joseph Kider (CS 15-466: Game Programming in Fall 2011)