GPGPU Lessons Learned

Mark Harris
General-Purpose Computation on GPUs

Highly parallel applications

- Physically-based simulation
- Image processing
- Scientific computing
- Computer vision
- Computational finance
- Medical imaging
- Bioinformatics
NVIDIA GPU Pixel Shader GFLOPS

• GPU Observed GFLOPS
• CPU Theoretical peak GFLOPS

- NV30
- NV35
- NV40
- G70
- G70-512
- G71

- 3.46 GHz Dual Core Pentium 4

Jan 2003 to May 2005 to Nov 2006
Physics on GPUs

GPU: very high data parallelism
- G70 24 pixel pipelines, 48 shading processors
- 1000s of simultaneous threads
- Very high memory bandwidth
- SLI enables 1-4 GPUs per system

Physics: very high data parallelism
- 1000s of colliding objects
- 1000s of collisions to resolve every frame

*Physics is an ideal match for GPUs*
Physically-based Simulation on GPUs

Particle Systems

Fluid Simulation

Cloth Simulation

Soft-body Simulation

Jens Krüger, TU-Munich

Doug L. James, CMU
What about Game Physics?

- Fluids, particles, cloth map naturally to GPUs
  - Highly parallel, independent data

- Game Physics = rigid body physics
  - Collision detection and response
  - Solving constraints

- Rigid body physics is more complicated
  - Arbitrary shapes
  - Arbitrary interactions and dependencies
  - Parallelism is harder to extract
Havok FX

A framework for Game Physics on the GPU
Joint NVIDIA / Havok R&D project launched in 2005

For details, come to the talks:

Havok FX™: GPU-accelerated Physics for PC Games
4:00-5:00PM Thursday
[Need location]

Physics Simulation on NVIDIA GPUs
5:30-6:30PM Thursday
[Need Location]
Havok FX Demo

NVIDIA DinoBones demo
Lessons learned from Havok FX

- Arithmetic Intensity is Key
- CPUs and GPUs can get along
- Readback ain’t wrong
- Vertex Scatter vs. Pixel Gather
- Printf debugging for pixel shaders
Arithmetic Intensity is Key

- Arithmetic Intensity = Arithmetic / Bandwidth

- GPUs like it high
  - Very little on-chip cache
  - Going to mem and back costs a lot
  - Long programs with much more math than texture fetch

- Game physics has very high AI
  - > 1500 pixel shader cycles per collision
  - ~ 100 texture fetches per collision
Havok GPU Threading Experiment

Performance of a pixel shader
Leverage Processor Strengths

- GPUs are good at data parallel computation
- CPUs are good at sequential computation

Most real problems have a bit of both
- Luckily most real computers have both processors!
- Especially game platforms

Rigid body collision processing is a great example
Rigid Body Dynamics Overview

- 3 phases to every simulation clock tick
  - Integrate positions and velocities
  - Detect collisions
  - Resolve collisions

Integration is embarrassingly parallel
- No dependencies between objects: use the GPU

Detecting collision is basically scene traversal
- CPU is good at this – use it

Resolving collisions is a tricky one
- Is it parallel enough for the GPU?
Is physics a data parallel task?

Solve Collisions

Body 1

Body 2

Body 4

Body 6

Body 7

Body 8

Body 3

Body 5

Contacts & Velocities

New Velocities
Is physics a data parallel task?

Solve Collisions

Contacts & Velocities

New Velocities
Is physics a data parallel task?

Solve Collisions

Contacts

Batch 1

Batch 2

Batch 3

New Velocities
Readback is Not Evil

- Hybrid CPU-GPU solution implies communication
  - Readback and download across PCI-e bus

- It’s not that bad if you use it wisely
  - Minimize transfer size and frequency
  - Use PBO to optimize transfers

- Physics data << computation
  - Read back and download a few bytes per obj each frame
  - At most a few MB per frame < 200 MB/sec
  - PCI-e = 4 GB / sec
Vertex Scatter vs. Pixel Gather

Problem: sparse array update
- Computed a set of addresses that need to be updated
- Compute updates for only those addresses in an array
Method 1: Pixel Gather

(Pre)compute an array of compute flags
Process all pixels in the destination array
Branch out of computation where flag is zero
Method 2: Vertex Scatter

- (Pre)compute addresses of elements to update
- Draw 1-pixel points at those addresses
  - Run update shader on points
Vertex Scatter vs. Pixel Gather

- Not obvious that Vertex Scatter can be a win
  - Drawing single-pixel points is inefficient
  - Because shader pipes process 2x2 “quads” of pixels

- But you can use a simple heuristic
  - Use Vertex Scatter if # of updates is significantly smaller than array size
  - Otherwise use pixel gather

- But always experiment in your own application!
Printf for Pixels

- Debugging pixel shaders is hard
  - Especially GPGPU shaders – output not an image
- Even harder if you’ve used all of your outputs
  - Havok FX easily uses up 4 float4 MRT outputs

- A simple hack to dump data from your shaders
  - A macro to dump arbitrary shader variables
  - A wrapper function to run the program once for all “printfs”
    - And run it once more with correct outputs
Printf for Pixels

First, define a handy macro to put in your shaders

```c
#ifdef DEBUG_SHADER
#define CG_PRINTF(index, variable) \  
    if (debugSelector == index) \  
        DEBUG_OUT = variable; \  
#else
#define CG_PRINTF(index, variable)
#endif
```
Printf for Pixels

```cpp
float4 foo( float2 coords, // other params
    #ifdef DEBUG_SHADER
        uniform float debugSelector
    #endif
    ) : COLOR0 {
    #ifdef DEBUG_SHADER
        float4 DEBUG_OUT;
    #endif
    float4 temp1 = complexCalc1();
    float4 temp2 = complexCalc1(temp1);
    float4 ret = complexCalc3(temp2);

    CG_PRINTF(1, temp1);
    CG_PRINTF(2, temp2);

    #ifdef DEBUG_SHADER
        CG_PRINTF(0, ret);
        return DEBUG_OUT;
    #else
        return ret;
    #endif
}
```

Use the macro to instrument your shader
Printf for Pixels: C++ code

def debugProgram( CGProgram prog, x, y, w, h, float** debugData)
{
    CGParam psel = cgGetNamedParameter(prog, "debugSelector");
    for (int selector = 1; selector < 100; ++selector) {
        if (debugData[selector] == 0) break;

        // run program with debug selector
        cgGLBindFloat1f(psel, selector);
        runProgram( prog, x, y, w, h);

        // read back results
        glReadPixels(x, y, w, h, GL_RGBA, GL_FLOAT,
                     debugData[selector]);
    }

cgGLBindFloat1f(psel, 0);
runProgram( prog, x, y, w, h); // run program as normal
}
Questions?

mharris@nvidia.com

Havok FX presentations at GDC 2006:

Havok FX™: GPU-accelerated Physics for PC Games
4:00-5:00PM Thursday
[Need location]

Physics Simulation on NVIDIA GPUs
5:30-6:30PM Thursday
[Need Location]