Comparison of CPU and GPGPU performance as applied to procedurally generating complex cave systems

Subject: Comp6470 - Special Topics in Computing
Student: Tony Oakden (U4750194)
Supervisor: Dr Eric McCreath
Introduction to Graphic Processing Units

Modern Graphics processing Units contain a great deal of computing power which can be leveraged in two ways:

1. As part of the graphics pipeline

2. Using a general purpose graphics programming API such as OpenCL or Cuda.

GPGPU programming has become a practical choice when solving problems which involve large numbers of calculations.
Objective of this project

- Reduce the time taken to create the mesh data for a procedurally generated cave system
- Investigate the relative benefits of different GPU optimization strategies
- Compare the performance of different hardware configurations
- Better understand OpenCL and Tessellation Shaders
The Cave creation algorithm

1. Start with one sphere

2. Add a random sphere which overlaps with at least one previously added sphere:
   - Random vector from previous sphere based on user defined parameters

3. Repeat step 2 till all spheres are added
Mesh creation algorithm

1. Find plane (portals) for each sphere intersection
2. Create mesh data for each sphere
3. Use portals to remove overlapping verts and triangles for each pair of spheres
4. Deform verts using portal data and suitable noise function
5. Weld verts between meshes at joints
Rendering

- Use previously calculated portals to create scene graph
- Traverse scene using classic “portal renderer” algorithm
- Combine all visible sub meshes in a single mesh and pass to GPU for rendering
Result
For now we only optimise step 5

- Step 4, mesh smoothing, is the most computationally intensive
- The Algorithm lends itself very well to GPGPU approach
- Quality scales roughly with number of verts in the donor spheres but so does the build time
Approach 1

• Rewrite mesh smoothing routine in OpenCL
  – Copy the data back to main memory after the GPU has finished
  – The rest of the process, including rendering, remains as it is in the original system
Approach 2

- Created a lower resolution version of the cave mesh

- Use the *tessellation shader* on the GPU to subdivide polygons and displace the vertices
Approach 3

• Use OpenCl to create a low resolution version of the cave, leaving the data on the GPU card

• Use the *tessellation shader* to increase the resolution in real-time

• Use *openCL, OpenGL* interoperability to pass data between the two stages
Expected Results

• Performance analysis of the 3 approaches including
  – different mesh complexities
  – comparison of the three different optimisation strategies
  – comparison of different hardware configurations

• Review of the OpenCL, Tessellation Shader technologies along with observations on the development process.

• Recommendation of the optimal approach and further work
Progress so far

• Mesh deformation routine successfully implemented in OpenCL

• Results from three different hardware platforms collected

• Greater understanding of OpenCL
First OpenCL experiments

Sphere: 496    Verts: 2,648,580

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPU</th>
<th>CPU Time</th>
<th>GPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>I7 2.4Ghz</td>
<td>GTX 860M</td>
<td>5.4</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>(1152)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I7 3.4Ghz</td>
<td>GTX 770</td>
<td>4.88</td>
<td>0.49</td>
</tr>
<tr>
<td></td>
<td>(1536)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I5 3.1Ghz</td>
<td>AMD 7800</td>
<td>6.2</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>(1280)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Next Step

• Implementation of tessellation shader
• Collect experimental data
• Analysis of results
• Further Research and report writing