General Purpose Computation on GPUs

Takeshi Hakamata
Outline

- Motivations
- GPGPU Programming Model
- Examples
Motivations

- GPUs are not limited to produce realistic images
- Linear Algebra, Computer Vision, Computational Geometry, etc.
Motivations

- Evolution of GPUs is much faster than that of CPUs
- Graphics is data parallel computation
  - Vertices and pixels

Use GPUs as co-processors.
First GPGPU applications

- In the mid 90s
- Fixed Function OpenGL Pipeline
  - Depth buffer (glDepthFunc)
  - Stencil buffer (glStencilOp)
  - Accumulation buffer (glAccum)
  - Logical operations (glLogicOp)
  - Alpha blending (glAlphaFunc)
First GPGPU applications

Game of Life

- Count the number of live neighbors using the stencil buffer
First GPGPU applications

Voronoi Diagram

- Render cones with their apexes at points
- Depth buffer does computation
Modern GPU features

- Programmable shaders
- Stream Processor [Khailany et al. 01]
  - Gather Operation
  - No Scatter Operation
  - SIMD semantics
    - Same program is executed over all elements
- Floating point framebuffer
GPU Programming Model

- Vertex and fragment processors
  - Most applications use the fragment shader
  - Vertex shader will be more popular
- Textures: arrays
- Texture fetch: read only memory operation
GPU Programming Model

- Render to Texture: write only memory operation
  - Use one of the followings:
    - Framebuffer Object (FBO) extension
    - RenderToTexture extension and a pbuffer
    - glCopyTexSubImage2D() and a pbuffer
“Ping Pong” buffering: iterations
- Use two framebuffers
- One is used for reading, the other is used for writing
- At the end of a loop, flip two buffers
GPU Programming Model

- Conditionals
  - Not well supported in the shader
  - Break a quad into several regions
A Typical Program

- Create floating point framebuffers using FBO or pbuffer
- Load textures
- Setup orthographic projection
  - 1 to 1 mapping between pixels and texels
- Render a viewport-sized quad
  - Input to the fragment program
- Do computation in a fragment program
- Render the result to texture
Example: Edge Detector

- Detect edges in a static image
- Using two Sobel kernels
Example: Edge Detector

- Load a texture image.
Example: Edge Detector

- Invoke computation by drawing a viewport-sized quad.

```c
glBegin(GL_QUADS);
    glTexCoord2f(0, 0); glVertex3f(-1, -1, -0.5f);
    glTexCoord2f(1, 0); glVertex3f( 1, -1, -0.5f);
    glTexCoord2f(1, 1); glVertex3f( 1,  1, -0.5f);
    glTexCoord2f(0, 1); glVertex3f(-1,  1, -0.5f);
glEnd();
```
Example: Edge Detector

For each pixel drawn, compute the convolution of Sobel kernel and the texture image.

```glsl
half4 main(vert2frag IN, uniform sampler2D tex) : COLOR {
    static const half offset = 1.0 / 512.0;
    half4 tile[9];

    tile[0] = tex2D(tex, IN.coords + half2(-offset,  offset));
    tile[1] = tex2D(tex, IN.coords + half2(      0,  offset));
    tile[2] = tex2D(tex, IN.coords + half2( offset,  offset));
    tile[3] = tex2D(tex, IN.coords + half2(-offset,       0));
    tile[4] = tex2D(tex, IN.coords);
    tile[5] = tex2D(tex, IN.coords + half2( offset,       0));
    tile[6] = tex2D(tex, IN.coords + half2(-offset, -offset));
    tile[7] = tex2D(tex, IN.coords + half2(      0, -offset));
    tile[8] = tex2D(tex, IN.coords + half2( offset, -offset));

}
```
Example: Edge Detector

- Take the greater convolution value and it will be a final pixel value.

```c
half convHLen = abs(convH.r) + abs(convH.g) + abs(convH.b);
half convVLen = abs(convV.r) + abs(convV.g) + abs(convV.b);
half pixel;

if (convHLen > convVLen) {
    pixel = convH / 4.0;
} else {
    pixel = convV / 4.0;
}

return pixel;
```
Example: Edge Detector

- Resulting image
Example: Reduction

- Reduce a texture map to a smaller one or even to a value

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

90
Example: Reduction

- Render points in odd-numbered locations
- Execute the fragment program

```cpp
Pixel = texture2D(tex, tc.st);
Pixel += texture2D(tex, tc.st + float2(1.0, 0.0));
Pixel += texture2D(tex, tc.st + float2(0.0, 1.0));
Pixel += texture2D(tex, tc.st + float2(1.0, 1.0));
```

- Halve the number of points in x and y
- Iterate
Example: Sparse Matrix Solver

- Using Conjugate Gradient method [Bolz et al. 03]
  - Sparse matrix-vector multiply
  - Vector inner product
- Iterative algorithm to solve $\mathbf{y} = \mathbf{A}\mathbf{x}$
- For physical simulation or optimization on unstructured meshes
Example: Sparse Matrix Solver

- Store off-diagonal non-zero entries in texture $A_j$.
- Store diagonal entries in texture $A_i$.
- Store the starting address of each row in a indirect texture $R$.
- $C$ stores “column number” of each entry.
- $X$ stores entries of the vector $x$. 
Example: Sparse Matrix Solver

$A_j^a$ -- off-diagonal matrix entries

$\mathcal{R}^x$ -- pointers to segments
Example: Sparse Matrix Solver

- $X^x$ - vertex positions
- $R^x$ - pointers to segments
- $A^x_i$ - diagonal matrix entries
- $A^x$ - off-diagonal matrix entries
- $C^a$ - pointers into $X$
Summary

- GPU is a very fast SIMD processor
- Often data packing in textures is the key to achieve good performance
- Not as easy to program on GPUs as on CPUs
  - Memory access
Information

- [http://www.gpgpu.org](http://www.gpgpu.org)
  - Tutorials
  - Programming Help
  - Knowledge Base