CS 380 - GPU and GPGPU Programming
Lecture 14: GPU Texturing 1

Markus Hadwiger, KAUST
Reading Assignments #8, #9 (until Oct 28)

Read (required):

• Programming Massively Parallel Processors book, 3rd edition (!), Chapter 7 (*Parallel Patterns: Convolution*)

• Interpolation for Polygon Texture Mapping and Shading, Paul Heckbert and Henry Moreton
  http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.7886

• MIP-Map Level Selection for Texture Mapping
  http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=765326

Read (optional):

• Vulkan Tutorial
  https://vulkan-tutorial.com
Quiz #2: Oct 23

Organization

• First 30 min of lecture
• No material (book, notes, ...) allowed

Content of questions

• Lectures (both actual lectures and slides)
• Reading assignments
• Programming assignments (algorithms, methods)
• Solve short practical examples
Choosing your own topic encouraged!
(we can also suggest some topics)

• Pick something that you think is really cool!
• Can be completely graphics or completely computation, or both combined
• Can be built on CS380 frameworks, NVIDIA OpenGL SDK, or CUDA SDK

Submit short (1-2 pages) project proposal sometime next week

• Send email or talk briefly with Peter or me before (!) you start writing your proposal to confirm that your plan is a suitable topic

Submit semester project and report (deadline: Dec 5)

Present semester project (we will schedule event in final exams week)
# Thread Blocks vs. Warps

<table>
<thead>
<tr>
<th>Technical Specifications</th>
<th>3.0</th>
<th>3.2</th>
<th>3.5</th>
<th>3.7</th>
<th>5.0</th>
<th>5.2</th>
<th>5.3</th>
<th>6.0</th>
<th>6.1</th>
<th>6.2</th>
<th>7.0</th>
<th>7.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum dimensionality of thread block</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum x- or y-dimension of a block</td>
<td></td>
<td>1024</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum z-dimension of a block</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum number of threads per block</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1024</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warp size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum number of resident blocks per multiprocessor</td>
<td>16</td>
<td></td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>Maximum number of resident warps per multiprocessor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>Maximum number of resident threads per multiprocessor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1024</td>
</tr>
<tr>
<td>Number of 32-bit registers per multiprocessor</td>
<td>64 K</td>
<td></td>
<td>128 K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>64 K</td>
<td></td>
</tr>
<tr>
<td>Maximum number of 32-bit registers per thread block</td>
<td>64 K</td>
<td>32 K</td>
<td>64 K</td>
<td></td>
<td>32 K</td>
<td></td>
<td>64 K</td>
<td></td>
<td></td>
<td></td>
<td>32 K</td>
<td>64 K</td>
</tr>
<tr>
<td>Maximum number of 32-bit registers per thread</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>63</td>
</tr>
<tr>
<td>Maximum amount of shared memory per multiprocessor</td>
<td>48 KB</td>
<td>112 KB</td>
<td>64 KB</td>
<td>96 KB</td>
<td>64 KB</td>
<td>96 KB</td>
<td>64 KB</td>
<td>96 KB</td>
<td>64 KB</td>
<td>96 KB</td>
<td>64 KB</td>
<td></td>
</tr>
<tr>
<td>Maximum amount of shared memory per thread block</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of shared memory banks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
__global__ void MatrixMul( float *matA, float *matB, float *matC, int w )
{
    __shared__ float blockA[ BLOCK_SIZE ][ BLOCK_SIZE ];
    __shared__ float blockB[ BLOCK_SIZE ][ BLOCK_SIZE ];

    int bx = blockIdx.x; int tx = threadIdx.x;
    int by = blockIdx.y; int ty = threadIdx.y;

    int col = bx * BLOCK_SIZE + tx;
    int row = by * BLOCK_SIZE + ty;

    float out = 0.0f;
    for ( int m = 0; m < w / BLOCK_SIZE; m++ ) {
        blockA[ ty ][ tx ] = matA[ row * w + m * BLOCK_SIZE + tx ];
        blockB[ ty ][ tx ] = matB[ col + ( m * BLOCK_SIZE + ty ) * w ];
        __syncthreads();
        for ( int k = 0; k < BLOCK_SIZE; k++ ) {
            out += blockA[ ty ][ k ] * blockB[ k ][ tx ];
        }
      __syncthreads();
    }

    matC[ row * w + col ] = out;
}
```

Caveat: for brevity, this code assumes matrix sizes are a multiple of the block size (either because they really are, or because padding is used; otherwise guard code would need to be added)
GPU Texturing

Rage / id Tech 5 (id Software)
Remember: Basic Shading

- Flat shading
  - compute light interaction per polygon
  - the whole polygon has the same color
- Gouraud shading
  - compute light interaction per vertex
  - interpolate the colors
- Phong shading
  - interpolate normals per pixel
- Remember: difference between
  - Phong Lighting Model
  - Phong Shading
Traditional OpenGL Lighting

- Phong lighting model at each vertex (glLight, …)
- Local model only (no shadows, radiosity, …)
- ambient + diffuse + specular (glMaterial!)

- Fixed function: Gouraud shading
  - Note: need to interpolate specular separately!
- Phong shading: evaluate Phong lighting model in fragment shader (per-fragment evaluation!)
Why Texturing?

- Idea: enhance visual appearance of surfaces by applying fine / high-resolution details
OpenGL Texture Mapping

- Basis for most real-time rendering effects
- Look and feel of a surface
- Definition:
  - A *regularly sampled function* that is mapped onto every *fragment* of a surface
  - Traditionally an image, but…
- Can hold arbitrary information
  - Textures become general data structures
  - Sampled and interpreted by fragment programs
  - Can render into textures → important!
Types of Textures

- Spatial layout
  - Cartesian grids: 1D, 2D, 3D, 2D_ARRAY, …
  - Cube maps, …

- Formats (too many), e.g. OpenGL
  - GL_LUMINANCE16_ALPHA16
  - GL_RGB8, GL_RGBA8, …: integer texture formats
  - GL_RGB16F, GL_RGBA32F, …: float texture formats
  - compressed formats, high dynamic range formats, …

- External (CPU) format vs. internal (GPU) format
  - OpenGL driver converts from external to internal
Texturing: General Approach

- Texture space \((u,v)\)
- Object space \((x_O,y_O,z_O)\)
- Image Space \((x_I,y_I)\)

Parametrization

Rendering (Projection etc.)

Texels
Texture Projectors

Where do texture coordinates come from?

- **Online**: texture matrix/texcoord generation
- **Offline**: manually (or by modeling program)

spherical  cylindrical  planar  natural
Texture Projectors

Where do texture coordinates come from?

- **Offline**: manual UV coordinates by DCC program
- **Note**: a modeling problem!
Texture Wrap Mode

- How to extend texture beyond the border?
- Border and repeat/clamp modes
- Arbitrary \((s,t,\ldots) \rightarrow [0,1] \times [0,1] \rightarrow [0,255] \times [0,255]\)

repeat  mirror/repeat  clamp  border

![Repeat Texture](image1)
![Mirror/Repeat Texture](image2)
![Clamp Texture](image3)
![Border Texture](image4)
2D Texture Mapping

For each fragment:
interpolate the texture coordinates
(barycentric)

Or:
Use arbitrary, computed coordinates

Texture-Lookup:
interpolate the texture data
(bi-linear)

Or:
Nearest-neighbor for “array lookup”
3D Texture Mapping

For each fragment:
interpolate the texture coordinates (barycentric)

Or:
Use arbitrary, computed coordinates

Texture-Lookup:
interpolate the texture data (tri-linear)

Or:
Nearest-neighbor for “array lookup”
Texture Mapping

2D (3D) Texture Space
  | Texture Transformation
2D Object Parameters
  | Parameterization
3D Object Space
  | Model Transformation
3D World Space
  | Viewing Transformation
3D Camera Space
  | Projection
2D Image Space

Kurt Akeley, Pat Hanrahan
Thank you.