CS 380 - GPU and GPGPU Programming
Lecture 10+11: Shading and Compute APIs 1+2

Markus Hadwiger, KAUST
Reading Assignment #6 (until March 19)

Read (required):

• Programming Massively Parallel Processors book, Chapter 4 (*CUDA Threads*)
Shading Example

Multiple layers: procedural shading, fetch textures, do lighting, ...

Base  Bruns  Circle  Marks  Cd  Cs

Courtesy Kekoa Proudfoot
Example: Per-Pixel(Fragment) Lighting

Simulating smooth surfaces by calculating illumination for each fragment
Example: specular highlights (Phong illumination/shading)

per-fragment evaluation  linear interp. from vertices
High-frequency surface detail from “normal map” (texture storing surface normals) that yields per-fragment normal vectors for shading
Normal Map Example: Quake 2 Model

Note: Gloss map defines where to apply specular

Final result!
Normal Map Example

Model by Piotr Slomowicz
Normal Map Example
Normal Map Example
Doom 3 (id Software, 2004)
Half-Life 2: The Lost Coast (Valve, 2005)
Unreal Engine 3 (Epic Games, 2006...)
Crysis (Crytek, 2007)
Shading Languages

Ancestors

• Offline shading: Pixar’s RenderMan, ...
• Real-time: Stanford Shading Language, ...

Programmable shading on GPUs

• Old assembly language OpenGL extensions / Direct3D
• NVIDIA Cg (compiles to low-level OpenGL or Direct3D shaders)
• DirectX HLSL (compiles to low-level Direct3D shaders)
• OpenGL shading language (GLSL)
• Write
  – Fragment (pixel) shaders
  – Vertex shaders
  – Geometry shaders
  – ...

15
High-Level Shading Languages

Syntax similar to C, plus vector variables and vector instructions:

GLSL: \texttt{vec4} v1; Cg/HLSL: \texttt{float4} v1; \texttt{// same as float v1[4] in C}
    \texttt{ivec3} v2; \texttt{int3} v2; \texttt{// same as int v2[3] in C}

Swizzling: \texttt{vec4} v3 = v1.xzzy;

Basic C, plus built-in functions for graphics, plus C math functions

- Functions like \texttt{sin()}, \texttt{pow()}, \texttt{exp()}, ...
- Functions like \texttt{length()}, \texttt{distance()}, \texttt{normalize()}, \texttt{dot()}, ...
- Functions like \texttt{clamp()}, \texttt{mix()}, ...
Programmable Processing Stages

GLSL 1.50 (OpenGL 3.2)
- Vertex shaders (run on vertex processors)
- Geometry shaders (run on geometry processors)
- Fragment shaders (run on fragment processors)

From the language / API perspective it is useful to consider separate types of processors, even when all of these shaders are in reality executed on identical processing cores on current GPUs!
Direct3D 10 Pipeline

New geometry shader stage:

- Vertex -> geometry -> pixel shaders
- Stream output after geometry shader

Courtesy David Blythe, Microsoft
Example: GeForce 8
Vertex Shader (1)

Process vertices and their attributes

- Position
- Color, texture coordinate(s), ...

GLSL 1.20
“Pass-through“ example

- Pass through per-vertex color
- Transform vertex position with OpenGL Model-View-Projection matrix

```glsl
#version 120

// Vertex Shader Main

void main(void) {
    // Pass vertex color to next stage
    gl_FrontColor = gl_Color;
    // Transform vertex position before passing it
    gl_Position = gl_ModelViewProjectionMatrix
                  * gl_Vertex;
}
```
Geometry Shader (1)

Process whole primitives

- Emit vertices
- Emit primitive(s)

GL_EXT_geometry_shader4
“Pass-through” example

- Pass through (emit) all vertices
- Pass through (emit) one primitive

```glsl
#extension GL_EXT_geometry_shader4: enable

// Geometry Shader Main
void main(void) {
    // Iterates over all vertices in the input primitive
    for (int i = 0; i < gl_VerticesIn; ++i) {
        // Pass color and position to next stage
        gl_FrontColor = gl_FrontColorIn[i];
        gl_Position = gl_PositionIn[i];
        // Done with this vertex
        EmitVertex();
    }
    // Done with the input primitive
    EndPrimitive();
}
```
Fragment Shader (1)

Process fragments

- Write one or more output fragments
- Use input color, texture coordinates, ...
- Compute shading, sample textures, ...
- Optionally discard fragment
- MRT: multiple render targets

GLSL 1.20
“Pass-through“ example

- Pass through interpolated color as fragment color

```c
// Fragment Shader Main
void main(void) {
  // Pass fragment color
  gl_FragColor = gl_Color;
}
```
Detailed Geometry

- Coarse base mesh
- Refine through subdivision
- High-frequency detail from displacement mapping
Simple OpenGL Program (1)

Simplest option for window and context management: GLUT
Can use GLEW for OpenGL extensions (not shown here; see programming assignment!)

```c
#include <GL/glut.h>

// C Main function

int main( int argc, char** argv ) {
    // GLUT Initialization
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA );
    glutInitWindowSize( 512, 512 );
    // Create OpenGL Window
    glutCreateWindow( "Simple Window" );
    init(); // non-GLUT initializations
    // Register callbacks
    glutReshapeFunc( reshape );
    glutDisplayFunc( display );
    glutKeyboardFunc( keyboard );
    glutMouseFunc( mouse );
    // Event Loop
    glutMainLoop();
    return 0;
}

/// The result is a window with 512x512 pixels
```
Init a (vertex) shader

```c
// OpenGL initialization calls for shaders
void initShader() {
    // Vertex Shader code source
    const GLchar* vsSource = {
        "#version 120\n"
        "void main(void) {\n"
        "    gl_FrontColor = gl_Color;\n"
        "    gl_Position = gl_ModelViewProjectionMatrix
               * gl_Vertex;\n"
        "}\n"
    }

    // Create program and vertex shader objects
    programObject = glCreateProgram();
    vtxShader = glCreateShader( GL_VERTEX_SHADER );
    // Assign the vertex shader source code
    glShaderSource( vtxShader, 1, &vsSource, NULL );
    // Compile the vertex shader
    glCompileShader( vtxShader );
    // Attach vertex shader to the GPU program
    glAttachShader( programObject, vtxShader );
    // Create an executable to run on the GPU
    glLinkProgram( programObject );
    // Install vertex shader as part of the pipeline
    glUseProgram( programObject );
}

/// The result is a vertex shader acting as a
/// simplified version of the fixed functionality
```
\[ \text{Illum} = \text{ambient} + \text{diffuse} + \text{specular} = Ka \times I + Kd \times I \times (\cos \theta) + Ks \times I \times \cos^n(\phi) \]
Phong Shader: Setup Steps

Step 1: Create Shaders
  Create handles to shaders

Step 2: Specify Shaders
  Load strings that contain shader source

Step 3: Compiling Shaders
  Actually compile source (check for errors)

Step 4: Creating Program Objects
  Program object controls the shaders

Step 5: Attach Shaders to Programs
  Attach shaders to program obj via handle

Step 6: Link Shaders to Programs
  Another step similar to attach

Step 7: Enable Program
  Finally, let GPU know shaders are ready
Phong Shader: App Setup

```c
GLhandleARB phongVS, phongkFS, phongProg; // handles to objects

// Step 1: Create a vertex & fragment shader object
phongVS = glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);
phongFS = glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);

// Step 2: Load source code strings into shaders
glShaderSourceARB(phongVS, 1, &phongVS_String, NULL);
glShaderSourceARB(phongFS, 1, &phongFS_String, NULL);

// Step 3: Compile the vertex, fragment shaders.
glCompileShaderARB(phongVS);
glCompileShaderARB(phongFS);

// Step 4: Create a program object
phongProg = glCreateProgramObjectARB();

// Step 5: Attach the two compiled shaders
glAttachObjectARB(phongProg, phongVS);
glAttachObjectARB(phongProg, phongFS);

// Step 6: Link the program object
glLinkProgramARB(phongProg);

// Step 7: Finally, install program object as part of current state
glUseProgramObjectARB(phongProg);
```
This Shader Does
- Gives eye space location for v
- Transform Surface Normal
- Transform Vertex Location

```glsl
varying vec3 N;
varying vec3 v;

void main(void)
{
    v = vec3(gl_ModelViewMatrix * gl_Vertex);
    N = normalize(gl_NormalMatrix * gl_Normal);

    gl_Position = gl_ModelViewProjectionMatrix * gl_vertex;

    (Update OpenGL Built-in Variable for Vertex Position)
}
```

Created For Use Within Frag Shader
varying vec3 N;
varying vec3 v;

void main (void)
{
  #define gl_FrontLightProduct[0] ambient;
  // we are in Eye Coordinates, so EyePos is (0,0,0)
  vec3 L = normalize(gl_LightSource[0].position.xyz - v);
  vec3 E = normalize(-v);
  vec3 R = normalize(-reflect(L,N));

  //calculate Ambient Term:
  vec4 Iamb = gl_FrontLightProduct[0].ambient;

  //calculate Diffuse Term:
  vec4 Idiff = gl_FrontLightProduct[0].diffuse * max(dot(N,L), 0.0);

  // calculate Specular Term:
  vec4 Ispec = gl_FrontLightProduct[0].specular
               * pow(max(dot(R,E),0.0), gl_FrontMaterial.shininess);

  // write Total Color:
  gl_FragColor = gl_FrontLightModelProduct.sceneColor + Iamb + Idiff + Ispec;
}
Thank you.