Lecture 10:
Part 1: Discussion of “SIMT” Abstraction
Part 2: Introduction to Shading

Kayvon Fatahalian
CMU 15-869: Graphics and Imaging Architectures (Fall 2011)
Today’s agenda

- Response to early course evaluation
- Demo: rendering pipeline visualization

Throughput core review
- SIMT vs. traditional SIMD (implicit vs. explicit SIMD)
  - Group exercise: implement SIMT

Shading introduction (next time: shading languages)
Demo: graphics pipeline visualization
Review: SIMD execution

- Define SIMD

- How is SIMD execution expressed by a program?

  An easy, high-level description of Larrabee’s explicit vector instruction set (in supplemental reading):
  
Review: NVIDIA’S “SIMT”

- Machine provides SPMD abstraction
  (SPMD = single program multiple data)

- What is the program?

- What is the data?
Assume: fictitious throughput processor

- Decodes one instruction per clock
- Instruction broadcast to all eight execution units
- Instructions manipulate contents of 32-bit (scalar) registers
  - e.g., floating point or integer operations

Let's implement a SIMT execution system! (whiteboard)
Shader 1: conditional

```cpp
sampler mySamp;
Texture2D<float3> myTex;

float4 fragmentShader(float3 norm, float2 st, float4 frontColor, float4 backColor) {
    float4 tmp;
    if (norm[2] < 0) // sidedness check
    {
        tmp = backColor;
    }
    else
    {
        tmp = frontColor;
        tmp *= myTex.sample(mySamp, st);
    }
    return tmp;
}
```
Shader 2: nested conditional

```cpp
sampler mySamp;
Texture2D<float3> myTex;

float4 fragmentShader(float3 norm, float2 st, float4 frontColor, float4 backColor)
{
    float4 tmp;
    if (norm[2] < 0) // sidedness check
    {
        tmp = backColor;
    }
    else
    {
        tmp = frontColor;
        if (fontColor == float4(1.0, 0.0, 0.0, 1.0))
        {
            tmp *= myTex.sample(mySamp, st);
        }
        else
        {
            tmp *= 0.5;
        }
    }
    return tmp;
}
```
Shader 3: while loop (homework)

```cpp
sampler mySamp;
Texture2D<float3> myTex;

float4 fragmentShader(float3 norm, float2 st, float4 frontColor, float4 backColor)
{
    float4 tmp;
    if (norm[2] < 0) // sidedness check
    {
        tmp = backColor;
    }
    else
    {
        tmp = frontColor;
        while (tmp[0] < tmp[1])
        {
            tmp[0] += 0.1;
        }
    }
    return tmp;
}
```
Shader 4: scalar branch

```c
sampler mySamp;
Texture2D<float3> myTex;
float myParam;
float myLoopBound;

float4 fragmentShader(float3 norm, float2 st, float4 frontColor, float4 backColor)
{
    float4 tmp;
    if (myParam < 0.5)
    {
        float scale = myParam * myParam;
        tmp = scale * frontColor;
    }
    else
    {
        tmp = backColor;
    }
    return tmp;
}
```
Optimize for “uniform control”

- Logic shared across all “lanes” need only be performed once
  - Must be known at compile time (compiler generates different instructions)

- Intel ISAs (LRBni, x86+SSE/AVX, etc.)
- AMD’s upcoming Graphics Core Next
Assume: fictitious throughput processor

- Now decode two instruction streams per clock
- What do we do?
Shading Introduction (or review)
The rendering equation

\[ i(x, x') = v(x, x') \left[ l(x, x') + \int r(x, x', x'') i(x', x'') \, dx'' \right] \]

Note: using notation from Hanrahan 90 (to match reading)
The rendering equation

\[ i(x, x') = v(x, x')[l(x, x') + \int r(x, x', x'')i(x', x'') \, dx''] \]

BRDF = bi-directional reflectance distribution function

Specifies faction of light from given incoming direction reflected in given outgoing direction
Example reflection functions

Ideal Specular
- Reflection Law
- Mirror

Ideal Diffuse
- Lambert’s Law
- Matte

Specular
- Glossy
- Directional diffuse

Slide credit Pat Hanrahan
Example materials

Plastic  Metal  Matte

Slide credit Pat Hanrahan
Images from Advanced Renderman [Apodaca and Gritz]
More materials

Slide credit Pat Hanrahan
Images from Matusik et al. SIGGRAPH 2003
Simplification

- All light sources are point sources
- Lights emit equally in all directions $i(x', x_{l_i}) = L_i$
- Only illumination of a surface comes directly from light sources

\[ i(x, x') = \sum_{i=0,1,2} L_i v(x', x_{l_i}) r(x, x', x_{l_i}) \]
More sophisticated lights

- **Attenuating light** (intensity falls off with distance: $1/R^2$)

- **Spot light** (does not emit equally in all directions)

- **Environment light** (not a point source: defines light from all directions)
Pre-programmable OpenGL

- `glLight(light_id, parameter_id, parameter_value)`
  - 10 parameters (e.g., ambient/diffuse/specular color, position, direction, attenuation coefficient)

- `glMaterial(face, parameter_id, parameter_value)`
  - Face specifies front or back facing geometry
  - Parameter examples (ambient/diffuse/specular reflectance, shininess)
  - Material value could be modulated by texture data

- Parameterized shading function evaluated at each vertex
  - Summation over all enabled lights
  - Resulting per-vertex color modulated by result of texturing
Shading languages

- Want to support diversity in materials
- Want to support diversity in lighting conditions
- Allow application to extend renderer by providing programmatic definition of the shading function