Advanced Deferred Shading
Efficient and Scalable Shading for Many Lights
Outline...

1. GPU Overview
2. Shading recap
3. Forward Shading
4. Deferred Shading
5. Tiled Deferred Shading
6. And more!
The Graphics Processing Unit
A Brief History

1. **NVIDIA GeForce 256**, 1999
   - Fixed function

   - Increasing Programmability

3. **Unified Shaders** – **AMD Xbox360**, 2005
   - General Programmable Cores

4. **CUDA**, 2007
   - C/C++, Integers, Memory Access

5. **OpenCL**, 2009
The GPU: Now and Future

- NVIDIA GeForce 400 series
  - Fermi, 2010
  - Virtual Functions
  - Unified Memory Addressing
- AMD Radeon 7900 series
  - Graphics Core Next, 2012
  - Much the same actually.
- CPU/GPU integration
  - AMD Fusion APU
  - Intel Sandy Bridge
GPU Performance

- Around 8x flops
- Growing fast!
- Around 6x bandwidth
- Not growing fast enough!

Theoretical TFlops

- GTX580 ~1.6 TFlops
- Intel CPU ~0.2 TFlops

Theoretical GB/s

- GTX580 ~192 GB/s
- CPU ~32 GB/s
GPU Architecture – How do they do it?

### CPU
- Few complex tasks
  - 1–2 thread/core
- Clever instruction exec
  - Branch prediction etc
- Lots of cache
- Narrow Memory Bus

### GPU
- Many simple tasks
  - ~48 threads/core
  - Wide SIMD (Many ALUs)
- Task switching
  - Latency hiding
- Wide Memory Bus
Shading

- Color and intensity at sample point
  - Here: sample point = fragment
- Diffuse contribution needs:
  - Light direction
  - Normal
  - Material diffuse color
- Specular shading needs:
  - Light direction
  - Normal
  - Eye direction
  - Material shininess
  - Material specular color
Fragment Shader Example (1/2)

- Only One Light!
  - Position
  - Color
- Material
  - Reflectances
  - Untextured
- Vertex data
  - Interpolated
- Result color
  - Output to frame buffer

```plaintext
uniform vec3 lightPos;
uniform vec3 lightColor;
uniform vec3 materialDiffuse;
uniform vec3 materialSpecular;
uniform float materialShininess;

in vec3 samplePos;
in vec3 sampleNormal;

out vec4 resultColor;

void main()
{
  // see next slide...
}
```
out vec4 resultColor;

void main()
{
    vec3 lightDir = normalize(lightPos - samplePos);
    vec3 diffuse = dot(sampleNormal, lightDir) * materialDiffuse * lightColor;
    vec3 viewDir = normalize(-samplePos);
    vec3 h = normalize(lightDir + viewDir);
    vec3 specular = pow(dot(h, normal), materialShininess) * materialSpecular * lightColor;
    resultColor = vec4(diffuse + specular, 1.0);
}
Forward Shading

- The traditional approach
  - Still most common for real time?
    - Shifting now...
  - Single pass
  - Shading in fragment shaders

- Geometry generates fragments
  - Each fragment is shaded
  - Result is merged into frame buffer
    - E.g. overwrite if nearest.
Forward Shading – Problems

- Overdraw
  - New fragments replaces old
  - Shading re-computed
  - And again...
  - Lots of waste
    - Especially for many lights.

- Light Assignment
  - What lights to use?
  - Usually per ‘object’
    - E.g. A few thousand triangles

- Shader management
  - Must support varying numbers of lights
**Overdraw**

- Brighter = more overdraw
  - Lots of waste
Which lights to use?
- Lights vs geometry
- Complex problem
  - 1000s of objects/batches.
  - 1000s of lights

Want minimal set of lights.
- (Geometrically) Small batches!

Want quick drawing
- Large batches!
- Good batching for GPU/API.
- Few material changes
### Summary: Forward Shading

**The good**
- Single Pass
- Low storage overhead
  - Single Frame Buffer
- Simple if only single / few lights
- MSAA works

**The Bad**
- Overdraw
- Light management
  - Difficult for many lights
- Shader management
- Batching coupled with lighting
Deferred Shading

Key idea
- Defer (i.e. wait with) light computations

First do a geometry pass
- Sample geometry as per normal
- But, no lighting
- Store geometry attributes per pixel

Then do a shading pass
- Use attributes from first pass.
- Process lights one at a time.
Deferred Shading

- Render Geometry to G-Buffers
  - G-Buffers store attributes,
    - e.g. color, normal, position.

- For each light
  - Draw Light Bounds
    - For each fragment
      - Read G-Buffers
      - Compute Shading
      - Add to frame buffer

- No overdraw
- Trivial light management
Render G-Buffers

- All attributes needed for shading e.g.
  - Color
  - Normal
  - Position(Z)
  - Specular
No Lights!
- Just the geometry

More Outputs
- to G-buffer
- 4 Screen sized textures

Position not stored
- Use depth buffer.
- Reconstruct using inverse projection

```cpp
uniform vec3 materialDiffuse;
uniform vec3 materialSpecular;
uniform float materialShininess;

in vec3 samplePos;
in vec3 sampleNormal;

out vec3 gBufferNormal;
out vec3 gBufferDiffuse;
out vec3 gBufferSpecular;
out float gBufferShininess;

void main()
{
    gBufferNormal = normal;
    gBufferDiffuse = materialDiffuse;
    gBufferSpecular =
        materialSpecular;
    gBufferShininess =
        materialShininess);
}```
Draw Light Bounds

- For each light
  - Draw Light Bounds
    - For each fragment
- Read G-Buffers
- Compute Shading
- Add to frame buffer
uniform vec3 lightPosition;
uniform vec3 lightColor;
uniform float lightRange;

void main()
{
  vec3 color = texelFetch(colorTex, gl_FragCoord.xy);
  vec3 specular = texelFetch(specularTex, gl_FragCoord.xy);
  vec3 normal = texelFetch(normalTex, gl_FragCoord.xy);
  vec3 position = fetchPosition(gl_FragCoord.xy);

  vec3 shading = doLight(position, normal, color, specular, lightPosition, lightColor, lightRange);

  resultColor = vec4(shading, 1.0);
}
Summary: Deferred Shading

The good

- Enables many lights
- Trivial light management
- Simple shader management
- No overdraw

The Bad

- Difficult to support transparency
- Large frame buffer
  - Especially with MSAA
- High memory bandwidth usage
Tiled Deferred Shading

- Key goal
  - Solve bandwidth issue
- Read G-Buffers once
The Bandwidth problem (1/3)

- New type of overdraw
  - Light overdraw

- N lights cover a certain pixel $\rightarrow$
  - N reads from the same G-Buffer location

- Deferred Shading

```
for each light
  for each covered pixel
    read G-Buffer
    compute shading
    write frame buffer
```
The Bandwidth problem (2/3)

- Deferred Shading, Light Loop

```c
for each light
  for each covered pixel
    read G-Buffer
    compute shading
    write frame buffer
```

- We would like to re-arrange

```c
for each pixel
  read G-Buffer
  for each affecting light
    compute shading
    write frame buffer
```
The Bandwidth problem (3/3)

- Requires
  - sequential access to lights for each pixel
- List per pixel prohibitive
  - Lots of data
  - Slow construction
- Share list in screen space tiles
  - E.g. 32 x 32 pixels
  - Simple construction
  - Little storage
  - Coherent access
Tiled Deferred Shading

- The Light Grid

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

The image on the right illustrates the grid with different shades representing different values.
Demo Time

- Lets look at those tiles...
Tiled Deferred Shading

1. Render Scene to G-Buffers
2. Build Light Grid
3. Full Screen Quad (or CUDA, or Compute Shaders)
   ◦ For each pixel
     • Fetch G-Buffer Data
     • Find Tile
     • Loop over lights and accumulate shading
     • Write shading
vec3 computeLight(vec3 position, vec3 normal, vec3 albedo, vec3 specular, float shininess)
{
    vec3 viewDir = -normalize(position);

    ivec2 l = ivec2(gl_FragCoord.xy) / CELL_DIM);

    int count = lightGrid[l.x][l.y].x;
    int offset = lightGrid[l.x][l.y].y;

    vec3 shading = vec3(0.0, 0.0, 0.0);

    for (int i = 0; i < count; ++i)
    {
        int lightId = texelFetch(tileDataTex, offset + i);
        shading += doLight(position, normal, albedo, specular, shininess, viewDir, lightId);
    }

    return shading;
}


```c
void main()
{
    ivec2 fragPos = ivec2(gl_FragCoord.xy);
    vec3 albedo = texelFetch(albedoTex, fragPos).xyz;
    vec4 specShine =
        texelFetch(specularShininessTex, fragPos);
    vec3 position = unProject(gl_FragCoord.xy,
        texelFetch(depthTex, fragPos));
    vec3 normal = texelFetch(normalTex, fragPos).xyz;

    vec3 viewDir = -normalize(position);

    gl_fragColor = computeLight(position, normal, albedo,
        specShine.xyz, viewDir, specShine.w, fragPos)
}
```
# Summary: Tiled Deferred Shading

<table>
<thead>
<tr>
<th>The good</th>
<th>The Bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low bandwidth usage</td>
<td>Difficult to support transparency</td>
</tr>
<tr>
<td>Enables many lights</td>
<td>Large frame buffer</td>
</tr>
<tr>
<td>Trivial light management</td>
<td>◦ Especially with MSAA</td>
</tr>
<tr>
<td>Simple shader management</td>
<td></td>
</tr>
<tr>
<td>No overdraw</td>
<td></td>
</tr>
</tbody>
</table>
Tiled Forward Shading

- **Key Idea**
  - Use tiles to solve light assignment
  - For forward shading

- **Forward as usual**
  - Shading in fragment shader
  - Access grid for list of lights

- **Benefits**
  - No G–Buffer
  - Transparency

- **Can easily combine!**
  - Tiled Deferred for opaque geometry
  - Tiled Forward for transparent
Tiled Forward Shading

1. Build Light Grid
2. Render Scene
   - For each fragment
     - Use geometry attributes
     - Find Tile
     - Loop over lights and accumulate shading
     - Write shading

Notes:
- + FSAA
- + Transparency
- + No G-Buffers
- – Overdraw
in vec3 normal;
in vec3 position;
in vec2 texCoord;

uniform vec3 specular;
uniform float shininess;
uniform float diffuse;

void main()
{
    ivec2 fragPos = ivec2(gl_FragCoord.xy);
    vec3 albedo = texture2D(albedoTex, texCoord) * diffuse;
    vec3 viewDir = -normalize(position);

    gl_fragColor = computeLight(position, normal, albedo,
                                specular, viewDir, shininess, fragPos);
}
## Summary: Tiled Forward Shading

<table>
<thead>
<tr>
<th>The good</th>
<th>The Bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low bandwidth usage</td>
<td>Overdraw is back</td>
</tr>
<tr>
<td>Enables many lights</td>
<td>Coherency worse</td>
</tr>
<tr>
<td>Trivial light management</td>
<td>◦ Looong shaders</td>
</tr>
<tr>
<td>Simpler shader management</td>
<td></td>
</tr>
<tr>
<td>MSAA</td>
<td></td>
</tr>
<tr>
<td>Transparency</td>
<td></td>
</tr>
<tr>
<td>No G–Buffers</td>
<td></td>
</tr>
</tbody>
</table>
## Technique Comparison

<table>
<thead>
<tr>
<th></th>
<th>Deferred</th>
<th>Tiled Deferred</th>
<th>Tiled Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Innermost loop</td>
<td>Pixels</td>
<td>Lights</td>
<td>Lights</td>
</tr>
<tr>
<td>Light data access pattern</td>
<td>Sequential</td>
<td>Random</td>
<td>Random</td>
</tr>
<tr>
<td>Pixel data access pattern</td>
<td>Random</td>
<td>Sequential</td>
<td>Sequential</td>
</tr>
<tr>
<td>Re-use Shadow Maps</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Shading Pass</td>
<td>Deferred</td>
<td>Deferred</td>
<td>Geometry</td>
</tr>
<tr>
<td>G-Buffers</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Overdraw of shading</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Transparency</td>
<td>Difficult</td>
<td>Simple&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Simple</td>
</tr>
<tr>
<td>Supporting FSAA</td>
<td>Difficult</td>
<td>Difficult</td>
<td>Trivial</td>
</tr>
<tr>
<td>Bandwidth Usage</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Light volume intersection</td>
<td>Per Pixel</td>
<td>Per Tile</td>
<td>Per Tile</td>
</tr>
</tbody>
</table>
Performance Comparison

- Frame times, GTX 280

![Graph showing frame times for Tiled Forward, Traditional Deferred, and Tiled Deferred methods.](image)
## Performance: Scaling

<table>
<thead>
<tr>
<th>GPU</th>
<th>GTX280</th>
<th>GTX480</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-Buffer Depth</td>
<td>16-bit</td>
<td>32-bit</td>
</tr>
<tr>
<td>TiledDeferred</td>
<td>15.7</td>
<td>17.2</td>
</tr>
<tr>
<td>min / max</td>
<td>7.05 / 33.3</td>
<td>8.82 / 34.8</td>
</tr>
<tr>
<td>TiledFwd</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>min / max</td>
<td>26.5 / 410</td>
<td></td>
</tr>
<tr>
<td>TiledFwd-PreZ</td>
<td>45.5</td>
<td></td>
</tr>
<tr>
<td>min / max</td>
<td>12.1 / 125</td>
<td></td>
</tr>
<tr>
<td>Deferred</td>
<td>38.3</td>
<td>82.1</td>
</tr>
<tr>
<td>min / max</td>
<td>14.0 / 90.0</td>
<td>27.7 / 197</td>
</tr>
<tr>
<td>DeferredStencil</td>
<td>18.4</td>
<td>28.3</td>
</tr>
<tr>
<td>min / max</td>
<td>8.1 / 39.1</td>
<td>12.4 / 64.8</td>
</tr>
</tbody>
</table>
Deferred Lighting
- Factor out specular and diffuse color
- G-Buffer only store normal and shininess
- Output diffuse and specular shading
- Second geometry pass which multiplies colors

Light PrePass
- Much the same
- But with monochromatic specular highlight

Similar performance as deferred
- Only improves constant factors.

Limits shading model even further
Optimizations

- Exclude Small Lights (on screen)
  - E.g. small = radius < a tile
  - Use standard deferred
Further Reading

- *Tiled Shading* – Olsson and Assarsson, JGT 2011
- *Bending the Graphics Pipeline*, Andersson, SIGGRAPH 2010 Course
- *Improved Tiled Shading* – Olsson and Billeter Somewhere 2012 ;(
Cheers

- You may now ask them questions.