CS 6958
LECTURE 5
TM ANALYSIS
SHADING

January 20, 2014
Clarification

- Avoid global variables

- **class/struct types will cause compiler to fail**
  - What I meant was global instances of objects
  - You can definitely define and use classes/structs
Pass By Reference

- Do this whenever possible
- Copying arguments is slow
  - Especially for large data types
- Passing by const reference keeps the original data safe
Issue Rate:
- Want this as high as possible

iCache conflicts
- Tracked separately from “resource” conflicts

thread*cycles of resource conflicts
- FU conflicts, L1, L2, DRAM

thread*cycles of data dependence
- Nothing you can do about this (for now)
- Includes RF conflicts
## Default Areas (square mm)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Area (sq mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPADD</td>
<td>.003</td>
</tr>
<tr>
<td>FPMIN</td>
<td>.00072</td>
</tr>
<tr>
<td>FPCMP</td>
<td>.00072</td>
</tr>
<tr>
<td>INTADD</td>
<td>.00066</td>
</tr>
<tr>
<td>FPMUL</td>
<td>.0165</td>
</tr>
<tr>
<td>INTMUL</td>
<td>.0117</td>
</tr>
<tr>
<td>FPINV</td>
<td>.112</td>
</tr>
<tr>
<td>CONV</td>
<td>.001814</td>
</tr>
<tr>
<td>BLT</td>
<td>.00066</td>
</tr>
<tr>
<td>BITWISE</td>
<td>.00066</td>
</tr>
</tbody>
</table>
Instruction Caches

- Instruction caches are actually “double pumped”
  - Each bank can service 2 requests every cycle

- Is N banks as good as N caches?
  - Is N caches reasonable?

- Is N banks \(> T \) threads useful?
Some of the useful functions (more as we need them)

- invsqrt(float f)
- sqrt(float f)
- min(float a, float b)
- max(float a, float b)
-GetXRes()
-GetYRes()
-GetFrameBuffer()
Ray tracer design

The major components in a ray tracer are:

- Camera (Pixels to Rays)
- Objects (Rays to intersection info)
- Materials (Intersection info and light to color)
- Lights
- Background (Rays to Color)

- All together: a Scene
Ray tracing algorithm

foreach frame
  foreach pixel
    foreach sample
      generate ray
      intersect ray with objects
      shade intersection point
<table>
<thead>
<tr>
<th>Step</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>foreach frame</td>
<td>Ignore for now</td>
</tr>
<tr>
<td>foreach pixel</td>
<td>Atomic Increment</td>
</tr>
<tr>
<td>foreach sample</td>
<td>Ignore for now</td>
</tr>
<tr>
<td>generate ray</td>
<td>Camera</td>
</tr>
<tr>
<td>intersect ray with objects</td>
<td>Spheres, more soon</td>
</tr>
<tr>
<td>shade intersection point</td>
<td>Material</td>
</tr>
</tbody>
</table>
Foreach pixel

Row-major order

Tiled

Progressive

Frameless rendering

Parallel
Atomic Increment

- atomicinc(0)
  - Atomically increments global register 0
  - All threads have access to this register

```c
for(int pix = atomicinc(0); pix < xres*yres; pix = atomicinc(0))
    int i = pix / xres;
    int j = pix % xres;
```

- Compare to
  ```c
  for(int i=0; i < xres; i++)
      for(int j=0; j < yres; j++)
  ```
Atomic Increment

```c
for(int pix = atomicInc(0); pix < xres*yres; pix = atomicInc(0))
    int i = pix / xres;
    int j = pix % xres;
```

Potential pixel assignments

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>
Cameras – Coming Soon

foreach frame
foreach pixel
foreach sample
generate ray

[Diagram of camera setup]
Find Closest Object

foreach frame
  foreach pixel
    foreach sample
      generate ray
      intersect ray with objects

$ t_{\text{near}} $
Shading (can get very complex)

foreach frame
  foreach pixel
    foreach sample
      generate ray
      intersect ray with objects
      shade intersection point

Pixel gets a color
Shading

- Path tracing (and other global techniques)
  - Consider light from all sources
- Starting simple
  - Consider light from direct source(s)
Color Multiplication

- $C1 \cdot C2 =$

- If white=$<1,1,1>$, red=$<1,0,0>$, green=$<0,1,0>$

- white $\cdot c = c$

- red $\cdot$ green $= \text{black}$

- Remember, colors in range [0 .. 1]
  - Can only get darker by reflecting off surfaces
Direct Light (and thus shadows)

- If there is line of sight from hit point to light source, add light’s contribution
  - Pixel color += object color * light color

- Else it is in shadow
  - Do nothing
First we need a vector from hit point to light
- \( P = \text{hit point} \)
- \( \text{LPos} = \text{location of light} \)
- \( L = \text{LPos} - P \)

Ray \( \text{shadowRay}(P, \text{normalize}(L)) \);
intersect \( \text{shadowRay} \) with scene objects
determine if anything blocking light
Computing Direct Light

- We don’t care about intersections behind the light!

\[ \text{max}_t = |L| \quad \leftarrow \text{before normalizing!} \]

If \(! \ (\text{hit} \&\& \ 0 < \text{hit}_t < \text{max}_t)\)  

Pixel color += object color * light color
Computing Direct Light

- Sometimes we don’t need to cast a ray
- Hit surface is on opposite side of light
- Angle between normal and $L > 90$
  - $\cos \theta = N \cdot \text{normalize}(L)$
  - $N = \text{surface normal direction}$

if($\cos \theta < 0$)
skip shadow ray
Flipped Normals

- Normals don’t always point the right way
  - Depends on how $N$ calculated
  - $\theta = \text{wrong angle!}$

- $V = \text{camera ray direction}$

if($V \cdot N > 0$)
  $N = -N$

Always flip normals to be on same side as incoming ray
Shadows
What's Wrong Here?
Numerical Precision

- Zoomed in: ideal
Numerical Precision

- Zoomed in: reality
  (numerical roundoff)

- Object casts shadow on itself

False intersection point!
Solution

- Offset shadow ray origin in normal direction
  - \( P += N \times \epsilon \)
  - \( \epsilon = \) some small number

- Epsilon depends on scene (1e-3 .. 1e-6)
Lambert’s Cosine Law

- Light reaching surface is proportional to projected visible area: \( \cos \theta \)
- \( \theta \) = angle between light and normal
Lambertian shading

- Comes from a “rough” surface (at microscopic level)
- Light that reaches the surface is reflected equally in all directions
Lambertian shading

- Color at surface: \( \cos \theta \times \text{lightColor} \)
  - \( \cos \theta = (N \cdot L) \)
  - (where \( N \) and \( L \) are unit vectors)
Direct Light
Ambient light

- With this mechanism, the light in a shadowed region is 0 (black).
- To avoid this, use “ambient” lighting.

- For each scene, define $K_d$, $K_a$
  - $K_d + K_a = 1$
- $K_d$ light comes from direct sources
- $K_a$ light comes from “ambient” sources
Ambient light

- **Pixel color =**
  - `objectColor * [(N \cdot L) \cdot lightColor \cdot Kd + ambientColor \cdot Ka]`

- **Define some ambientColor for the scene**
  - Based on how bright scene is, background, etc…
  - Artistic choice (since it is a hack)
Direct + Ambient Light

- Ambient = <0.6, 0.6, 0.6>
- Kd = 0.7
- Ka = 0.3