Announcements

• Project 3 RELEASED
  • Due Wed, 10/8
• Office Hours on 10/5, 10/8 and 10/12
  • Out of town, will be looking on Google forums
  • Be specific about your questions!
Outline

• Motivation
• Review and basics
• Rendering equation
• Naïve pathtracing
• Parallelization
• Optimization
• Base Code Tour
Acknowledgements

- Karl Yining Li
- Liam Boone
Motivation
Motivation
RENDERING 101
Rendering 101: Human Perception

- Human vision: what we “see” is reflected light
- Light scattered from objects
Rendering 101: Human Perception

- Different materials have different scattering properties
Rendering 101 : Physics

• Light : wave-particle duality
  • Photons
  • Waves
• Energy
Rendering 101 : Rays

• Model light as rays
  • Assume light moves relatively straight
  • When hitting an object, the direction of the ray is modified
Rendering 101: Global Illumination

- Illumination not only from direct sources
Rendering 101 : Global Illumination

• Direct Illumination ONLY
Rendering 101: Global Illumination
Rendering 101

- “Light simulation”
- Costly to simulate from light’s perspective
  - Therefore, rays shot from camera
RENDERING EQUATION
Rendering Equation


\[ L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{S^2} f(p, \omega_o, \omega_i) L_d(p, \omega_i) |\cos \theta_i| d\omega_i \]
Rendering Equation : Diffuse

• Equal probability of scattering in all directions
Rendering Equation : Specular

• Perfect :
  • Ray reflected about the normal
Rendering Equation: Dielectrics

- Fresnel reflection / refraction
  - Shlick’s approximation
Algorithm 3 Path Tracing Main Loop

1: for each pixel (i,j) do
2:   Vec3 $C = 0$
3:   for (k=0; k < samplesPerPixel; k++) do
4:     Create random ray in pixel:
5:     Choose random point on lens $P_{lens}$
6:     Choose random point on image plane $P_{image}$
7:     $D = \text{normalize}(P_{image} - P_{lens})$
8:     Ray ray = Ray($P_{lens}$, $D$)
9:     castRay(ray, isect)
10:    if the ray hits something then
11:      $C' += \text{radiance}(\text{ray, isect, 0})$
12:    else
13:      $C' += \text{backgroundColor}(D)$
14:    end if
15:  end for
16:  image(i,j) = $C' / \text{samplesPerPixel}$
17: end for
PARALLELIZATION
GPU Pathtracers

- Peter and Karl’s GPU Path Tracer:
  - https://vimeo.com/41109177
- BRIGADE Renderer:
  - http://www.youtube.com/watch?v=FJLy-ci-RyY
- Octane otoy:
How many bounces per path?
How many bounces per path?
4?
How many bounces per path?
4? 3?
How many bounces per path?
4? 3? 2?
How many bounces per path?
4? 3? 2? 1?
How many bounces per path?
4? 3? 2? 1? 0?
How many bounces per ray?

- We have no idea!
- What does this imply about parallelizing by pixel?
Wasted Cycles

- GPU can only handle finite number of blocks at a time
- If some threads need to trace more bounces, then others might spend too much time idling

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
<th>Thread 4</th>
<th>Thread 5</th>
<th>Thread 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounce 1</td>
<td>Bounce 1</td>
<td>DONE</td>
<td>Bounce 1</td>
<td>Bounce 1</td>
<td>Bounce 1</td>
</tr>
<tr>
<td>Bounce 2</td>
<td>DONE</td>
<td></td>
<td>Bounce 2</td>
<td>DONE</td>
<td>DONE</td>
</tr>
<tr>
<td>Bounce 3</td>
<td></td>
<td></td>
<td>DONE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bounce 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WASTED CYCLES
Ray Parallelization

- Parallelize by ray, not pixel
- Multiple kernel launches that trace individual bounces

1. Construct pool of rays that need to be tested
2. Construct accumulator image, initialize black
3. Launch a kernel to trace ONE bounce
4. Add any new rays to the pool
5. Remove terminated rays from the pool
6. Repeat from 3 until pool is dry
Ray Parallelization

- Each iteration will have fewer rays, requiring fewer blocks, and giving faster execution.
- Works very well (even in practice)
- Be careful of edge cases!
Example

- Ray pool:
  - 1, 2, 3
- Threads needed:
  - 3
- Ray 1 terminates
Example

- Ray pool:
  - 2, 3
- Threads needed:
  - 2
Example

• Ray pool:
  • 2, 3

• Threads needed:
  • 2

• Ray 3 terminates
Example

• Ray pool:
  • 2

• Threads needed:
  • 1

• Ray 2 terminates
Optimization

- Wasted cycles / Zero contribution rays
  - Stream compaction!
- Acceleration structures
BASE CODE TOUR
Base Code

- Scene reader
- UI / Rendering visualization
  - PBO transfer
- Intersection code
- CPU image write
Project Expectations

• GPU Path tracing
  • Intersections, accumulation, etc. on GPU

• Diffuse and specular materials

• 2 of the following features:
  • Fresnel reflection / refraction
  • OBJ loading
  • Acceleration structure (BVH, k-d tree, etc.)
  • Depth of field
  • Subsurface scattering
  • Etc. (If you have other ideas, be sure to contact us)
Project Expectations

• Analysis for every extra feature:
  • Overview write up
  • When adding this feature, what is the performance impact? Why?
  • If there are special cases, why you implemented it the way you did.
  • How can this be optimized any further?
  • Pros / cons of GPU vs. CPU implementation
Tips for README

• Sell your project
• Assume reader has base knowledge of path tracing
• View this as a wiki to document your code
• Do NOT leave this for the last minute
Further Reading

• Physically Based Rendering, 2e. (Pharr, Humphreys)
• Realistic Image Synthesis Using Photon Mapping (Jensen)
• smallPT