Lecture 32 of 41

Lab 6: Ray Tracing
with ACM SIGGRAPH Demo & POV-Ray

William H. Hsu
Department of Computing and Information Sciences, KSU

Public mirror web site: http://www.kddresearch.org/Courses/CIS636
Instructor home page: http://www.cis.ksu.edu/~bhsu

Readings:

Last class: Chapter 14, Eberly 2e – see http://bit.ly/ieUq45
Today: Ray Tracing Handout
Next class: Chapter 15, Ray Tracing Handout
Lecture Outline

- Reading for Last Class: Chapter 14, Eberly 2e
- Reading for Today: Ray Tracing Handout
- Reading for Next Class: Chapter 15, Eberly 2e; Ray Tracing Handout
- Last Time: Ray Tracing (RT), Part 1 of 2
  - Vectors: Light (L) & shadow, Reflected (R), Transmitted & refraction
  - Basic recursive ray tracing & ray trees
  - Phong illumination model, texture mapping revisited
  - Distributed RT: survey, supersampling illustrated
  - Things you get “for free”: clipping, VSD (backface/occlusion culling)
- Today: Ray Tracing Lab
  - POV-Ray: http://www.povray.org
- Next Class: Ray Tracing 2 of 2
  - Hybridizing RT with radiosity (photon maps)
  - Progressive refinement
### Where We Are

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Topic</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Lab 4a: Animation Basics</td>
<td>Flash animation handout</td>
</tr>
<tr>
<td>22</td>
<td>Animation 2: Rotations, Dynamics, Kinematics</td>
<td>Chapter 17, esp. §17.1 – 17.2</td>
</tr>
<tr>
<td>23</td>
<td>Demos 4: Modeling &amp; Simulation, Rotations</td>
<td>Chapter 10, 13, §17.3 – 17.5</td>
</tr>
<tr>
<td>24</td>
<td>Collisions 1: axes, OBBS, Lab 4b</td>
<td>§2.4.3, 8.1, GL handout</td>
</tr>
<tr>
<td>25</td>
<td>Spatial Sorting: Binary Space Partitioning</td>
<td>Chapter 6, esp. §6.1</td>
</tr>
<tr>
<td>26</td>
<td>Demos 5: More CGA: Picking; HW/Exam</td>
<td>Chapter 7, § 8.4</td>
</tr>
<tr>
<td>27</td>
<td>Lab 5a: Interaction Handling</td>
<td>§ 8.3 – 8.4; 4.2, 5.0, 5.6, 9.1</td>
</tr>
<tr>
<td>28</td>
<td>Collisions 2: Dynamic, Particle Systems</td>
<td>§ 9.1, particle system handout</td>
</tr>
<tr>
<td>29</td>
<td>Exam 2 review: Hour Exam 2 (evening)</td>
<td>Chapters 5 – 6, 7 – 8, 12, 17</td>
</tr>
<tr>
<td>30</td>
<td>Lab 5b: Particle Systems</td>
<td>Particle system handout</td>
</tr>
<tr>
<td>31</td>
<td>Animation 3: Control &amp; IK</td>
<td>§ 8.3, CGA handout</td>
</tr>
<tr>
<td>32</td>
<td>Ray Tracing 1: intersections, ray trees</td>
<td>Chapter 14</td>
</tr>
<tr>
<td>33</td>
<td>Lab 6a: Ray Tracing Basics with POVRay</td>
<td>RT handout</td>
</tr>
<tr>
<td>34</td>
<td>Ray Tracing 2: advanced topic survey</td>
<td>Chapter 15, RT handout</td>
</tr>
<tr>
<td>35</td>
<td>Visualization 1: Data (Quantities &amp; Evidence)</td>
<td>Tufte handout (1)</td>
</tr>
<tr>
<td>36</td>
<td>Visualization 2: Objects</td>
<td>RT handout (2 &amp; 4)</td>
</tr>
<tr>
<td>37</td>
<td>Color Basics; Term Project Prep</td>
<td>Color handout</td>
</tr>
<tr>
<td>38</td>
<td>Lab 7: Fractals &amp; Terrain Generation</td>
<td>Fractals/Terrain handout</td>
</tr>
<tr>
<td>39</td>
<td>Visualization 3: Processes; Final Review 1</td>
<td>Tufte handout (3)</td>
</tr>
<tr>
<td>40</td>
<td>Project presentations 1: Final Review 2</td>
<td>–</td>
</tr>
<tr>
<td>41</td>
<td>Project presentations 2</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Final Exam</td>
<td>Ch. 1 – 8, 10 – 15, 17, 20</td>
</tr>
</tbody>
</table>

Lightly-shaded entries denote the due date of a written problem set, heavily-shaded entries, that of a machine problem (programming assignment), blue-shaded entries, that of a paper review, and the green-shaded entry, that of the term project.

Green, blue and red letters denote exam review, exam, and exam solution review dates.
Acknowledgements:
Ray Tracing

Dave Shreiner & Brad Grantham
Adjunct Professor & Adjunct Lecturer,
Santa Clara University
ARM Holdings, plc
http://www.plunk.org/~shreiner/
http://www.plunk.org/~grantham/

David K. Buck, Aaron Collins, et al.
Developers
Persistence of Vision Raytracer (POV-Ray)
http://www.povray.org

G. Scott Owen & Yan Liu
Professor Emeritus / ACM SIGGRAPH President &
Graduate Research Assistant
Hypermedia and Visualization Laboratory
Department of Computer Science
Georgia State University / ACM
http://www.cs.gsu.edu/gsowen/
Review [1]:
Reasons for Using Ray Tracing

- Simulate rays of light
- Produces natural lighting effects
  - Reflection
  - Refraction
  - Soft Shadows
  - Depth of Field
  - Motion Blur
  - Caustics

- Hard to simulate effects with rasterization techniques (OpenGL)
- Rasterizers require many passes
- Ray-tracing easier to implement

Adapted from slides by 2001 D. Shreiner & B. Grantham, SCU
Review [2]:
How Ray Tracing Works

- Trace rays from eye instead
- Do work where it matters

This is what most people mean by “ray tracing”.

Adapted from slides © 2001 D. Shreiner & B. Grantham, SCU
Want to know: at what point $p$ does ray intersect triangle?

Compute lighting, reflected rays, shadowing from that point $p = t_{\min}$

$r_0$, $r_d$, $<?, ?, ?>$ (t = ???)
We’ll use triangles for lights
Can build complex shapes from triangles
Some lighting terms

Review [4]:
General Notation Review

Adapted from slides ©2001 D. Shreiner & B. Grantham, SCU
Recursive ray evaluation

```c
rayTrace(ray) {
    hitObject(ray, p, n, triangle);
    color = object color;
    if(object is light)
        return(color);
    else
        return(lighting(p, n, color));
}
```

Generates ray tree shown at right
Calculating surface color

```c
lighting(point) {
    color = ambient color;
    for each light
        if(hitObject(shadow ray))
            color += lightcolor * 
            dot(shadow ray, n);
    color += rayTrace(reflection) * 
    pow(dot(reflection, ray), shininess);
    return(color);
}
```
**Review [7]: More Quality, More Speed**

- Better Lighting + Forward Tracing
- Texture Mapping
- Modeling Techniques
- Distributed Ray Tracing: Techniques
  - Motion Blur
  - Depth of Field
  - Blurry Reflection/Refraction
- Improving Image Quality
- Acceleration Techniques

Adapted from slides © 2001 D. Shreiner & B. Grantham, SCU
Review [8]: Distributed Ray Tracing
Review [9]:
Supersampling, “Forward” RT

- One ray is not enough (jaggies)
- Can use multiple rays per pixel - **supersampling**
- Can use a few samples, continue if they’re very different - **adaptive supersampling**
- Texture interpolation & filtering

“Forward” RT for Caustics

http://bit.ly/dV7I8m

Adapted from slides ♥ 2001 D. Shreiner & B. Grantham, SCU
Lab 6a [1]:
ACM SIGGRAPH 2-D RT Program Help

This Java program was written by Yan Liu, under the supervision of Dr. G. Scott Owen, Department of Computer Science at Georgia State University. It is based on a Pascal program written by Olimo Schweitzer. All Copyrights are Reserved by Dr. G. Scott Owen.

Solid Sphere

Transparent Sphere

Demonstration of the Ray Trace Algorithm
This is a demonstration of the raytracing rendering algorithm. A simple 2-D environment is shown. The user can arrange four predefined objects within the environment (see description on the left) and raytrace the result. The raytrace algorithm forms and intersects primary and secondary rays to compute the final shade at each pixel in the frame buffer.

Polygon

Light Source

Please choose 'move' to get started. Then you can move the objects around. Then choose 'trace' to trace the rays.

Screenshots from Java program © 2001 G. S. Owen & Y. Liu, GSU

CIS 536/636
Introduction to Computer Graphics
Lecture 32 of 41
Computing & Information Sciences
Kansas State University
Lab 6a [2]:
Trace Screen

MAIN ALGORITHM
For each pixel
- Form primary ray
- Find closest intersection
- Shade (DEPTH+1, final)
- Put final shade in pixel

SCREEN
EYE
SHADE (DEPTH, RTNSHADE)

Form shadow ray
Find intersection
If reflective
- Form reflect ray
- Find closest intersect
- Shade (DEPTH+1, REFNSHADE)
If transparent
- Form refract ray
- Find closest intersect
- Shade (DEPTH+1, REFRNSHADE)
Compute rnsshade

Screenshots from Java program © 2001 G. S. Owen & Y. Liu, GSU

CIS 536/636
Introduction to Computer Graphics
Lecture 32 of 41
Computing & Information Sciences
Kansas State University
Lab 6a [3]:
First Ray (Click “Clear” & “Auto”)

MAIN ALGORITHM
For each pixel
  Form primary ray
  Find closest intersection
  If intersect something
    Shade (DEPTH=1, final)
  Put final shade in pixel

SHADE (DEPTH, RTNSHADE)
Form shadow ray
Find intersection
If reflective
  Form reflect ray
  Find closest intersect
  Shade (DEPTH+1, REFNSHADE)
If transparent
  Form refract ray
  Find closest intersect
  Shade (DEPTH+1, REFNSHADE)
Compute refract
Lab 6a [4]: Second Ray (Click “Auto” to Advance)

MAIN ALGORITHM

For each pixel
Form primary ray
Find closest intersection
If intersect something
Shade (DEPTH+1, final)
Put final shade in pixel

SCREEN

EYE

SHADE (DEPTH, RTHSHADE)

Form shadow ray
Find intersection
If reflective
Form reflect ray
Find closest intersect
Shade (DEPTH+1, REFISHADE)
If transparent
Form refract ray
Find closest intersect
Shade (DEPTH+1, REFRESHADE)
Compute the shade
Lab 6a [5]: Third Ray

MAIN ALGORITHM

For each pixel
   Form primary ray
   Find closest intersection
   If transparent something
      Shade (DEPTH+1, final)
   Put final shade in pixel

SHADE (DEPTH, RTNISSHADE)

Form shadow ray
Find intersection
If reflective
   Form reflect ray
   Find closest intersect
   Shade (DEPTH+1, REFRESHADE)
   If transparent
   Form refract ray
   Find closest intersect
   Shade (DEPTH+1, REFRESHADE)
   Compute the shade

Screenshots from Java program © 2001 G. S. Owen & Y. Liu, GSU
Lab 6a [6]: Fourth Ray

MAIN ALGORITHM

For each pixel
- Form primary ray
- Find closest intersection
- If intersect something Shade (DEPTH+1, final)
- Put final shade in pixel

SCREEN

EYE

SHADE (DEPTH, RTN SHADE)

Form shadow ray
Find intersection
Reflective
- Form reflect ray
- Find closest intersect Shade (DEPTH+1, REF SHADE)
- Transparent
- Form reflec ray
- Find closest intersect Shade (DEPTH+1, REF SHADE)
- Compute-ref shade

Screenshots from Java program © 2001 G. S. Owen & Y. Liu, GSU
Lab 6a [7]: Fifth Ray

**MAIN ALGORITHM**

For each pixel:
- Form primary ray
- Find closest intersection
- If intersected
  - Shade (DEPTH+1, final)
- Put final shade in pixel

**SHADE (DEPTH, RTN SHADE)**

Form shadow ray
Find intersection
- If reflective:
  - Form reflect ray
  - Find closest intersect
  - Shade (DEPTH+1, REF SHADE)
  - Transparent:
  - Form reflect ray
  - Find closest intersect
  - Shade (DEPTH+1, REF SHADE)
  - Compute rhinshade
Lab 6b [1]:
POV-Ray

"Office" © 2004 Jaime Vives Piqueres

"My First CGSphere" © 2008 Robert McGregor
Lab 6b [2]:
POV-Ray

"The Wet Bird" © 2001 Gilles Tran
http://bit.ly/gMBuGt

"Dissolution" © 2005 Newt

"Thanks for all the fish" © 2008 Robert McGregor

Images © respective authors, generated using POV-Ray
Summary

- Reading for Last Class: Chapter 14, Eberly 2e
- Reading for Today: Ray Tracing Handout
- Reading for Next Class: Chapter 15, Eberly 2e; Ray Tracing Handout
- Last Time: Ray Tracing (RT), Part 1 of 2
  - Vectors: I (incident ray), L, R, T
  - Basic recursive ray tracing & ray trees
  - Distributed RT: survey, supersampling illustrated
- Today: Ray Tracing Lab
    - 2-D “screen”
    - Moveable objects: transparent, opaque (both reflective)
  - POV-Ray ([http://www.povray.org](http://www.povray.org)) Example Renderings
- Next Class: Ray Tracing 2 of 2
  - Progressive refinement radiosity (photon maps) introduced
  - Using RT/radiosity together and with shading
Terminology

- **Ray Tracing aka Ray Casting**
  - Given: screen with pixels \((u, v)\)
  - Find intersection \(t_{\min}(u, v)\) of rays through each \((u, v)\) with scene
  - Vectors emanating from world-space coordinate of \(t_{\min}\)
    - **Light** (L) aka **Source** (S): to point light sources (or shadows)
    - **Reflected** (R): from object surface
    - **Transmitted** or **Transparency** (T): through transparent object
  - **Recursive RT**: call raytracer for each intersection, get ray tree
  - **Incident vector** (I): incoming from eye

- **Caustic**: Envelope of Light Rays Reflected/Refracted by Curved Object
  - Example: Slide 13 (today’s lecture)

- “Backward” RT: Eye-to-Scene, Scene-to-Light (Typical Order)
- “Forward” RT: Light-to-Scene, Scene-to-Eye (Only for Caustics)
- **Screen**: Parallel to View “Plane”, Rays Shot Through It