Lecture 31of 41

Ray Tracing, Part 1 of 2: Intersections, Ray Trees & Recursion

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

KSOL course pages: http://bit.ly/hGvXlH / http://bit.ly/eVizrE
Public mirror web site: http://www.kddresearch.org/Courses/CIS636
Instructor home page: http://www.cis.ksu.edu/~bhsu

Readings:
Last class: §5.3, Eberly 2e – see http://bit.ly/ieUq45; CGA Handout
Today: Chapter 14, Eberly 2e
Next class: Ray Tracing Handout
Reference – Wikipedia, *Ray Tracing*: http://bit.ly/dV7INm
Reference – ACM *Ray Tracing News*: http://bit.ly/fqyZNQ

Lecture Outline

- Reading for Last Class: §5.3, Eberly 2e; CGA Handout
- Reading for Today: Chapter 14, Eberly 2e
- Reading for Next Class: Ray Tracing Handout
- Last Time: Animation Part 3 of 3 – Inverse Kinematics
  - FK *vs.* IK
  - IK
    - Autonomous agents *vs.* hand-animated movement
    - Analytical vs. iterative solutions
  - Rag doll physics, rigid-body dynamics, physically-based models
- End of Material on: Particle Systems, Collisions, CGA, PBM
- Today: Ray Tracing, Part 1 of 2
  - Vectors: Light/shadow (L), Reflected (R), Transmitted/refracted (T)
  - Basic recursive ray tracing: ray trees
- Next Class: Ray Tracing Lab

Where We Are

21	Lab 4a: Animation Basics	Flash animation handout
22	Animation 2: Rotations; Dynamics, Kinematics	Chapter 17, esp. §17.1 – 17.2
23	Demos 4: Modeling & Simulation; Rotations	Chapter 10[1], 13[2], §17.3 – 17.5
24	Collisions 1: axes, OBBs, Lab 4b	§2.4.3, 8.1, GL handout
25	Spatial Sorting: Binary Space Partitioning	Chapter 6, esp. §6.1
26	Demos 5: More CGA; Picking; HW/Exam	Chapter 7[2]; § 8.4
27	Lab 5a: Interaction Handling	§ 8.3 – 8.4; 4.2, 5.0, 5.6, 9.1
28	Collisions 2: Dynamic, Particle Systems	§ 9.1, particle system handout
	Exam 2 review; Hour Exam 2 (evening)	Chapters 5 – 6, 7[2] – 8, 12, 17
29	Lab 5b: Particle Systems	Particle system handout
30	Animation 3: Control & IK	§ 5.3, CGA handout
31	Ray Tracing 1: intersections, ray trees	Chapter 14
32	Lab 6a: Ray Tracing Basics with POV-Ray	RT handout
33	Ray Tracing 2: advanced topic survey	Chapter 15, RT handout
34	Visualization 1: Data (Quantities & Evidence)	Tufte handout (1)
35	Lab 6b: More Ray Tracing	RT handout
36	Visualization 2: Objects	Tufte handout (2 & 4)
37	Color Basics; Term Project Prep	Color handout
38	Lab 7: Fractals & Terrain Generation	Fractals/Terrain handout
39	Visualization 3: Processes; Final Review 1	Tufte handout (3)
40	Project presentations 1; Final Review 2	–
41	Project presentations 2	–
	Final Exam	Ch. 1 – 8, 10 – 15, 17, 20

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: Inverse Kinematics, Ray Tracing

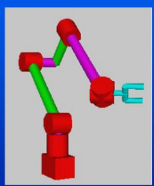
Dave C. Brogan
Visiting Assistant Professor, Computer Science Department, University of Virginia
http://www.cs.virginia.edu/~dbrogan/
Susquehanna International Group (SIG)
http://www.sig.com

Renata Melamud
Ph.D. Candidate
Mechanical Engineering Department
Stanford University
http://micromachine.stanford.edu/~rmelamud/

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

Review [1]: Kinematics & Degrees of Freedom

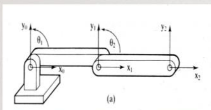
- How about this robot arm?



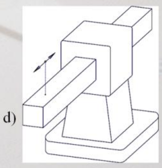
- Six again
  - 2-base, 1-shoulder, 1-elbow, 2-wrist

Adapted from slides © 2000 – 2005 D. Brogan, University of Virginia
CS 551, Advanced CG & Animation – http://bit.ly/hUXrqd

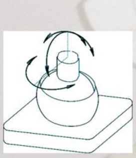
Review [2]: Joint Types



Revolute Joint
1 DOF (Variable - Y)



Prismatic Joint
1 DOF (linear) (Variables - d)



Spherical Joint
3 DOF (Variables - Y_1, Y_2, Y_3)

Adapted from slides © 2002 R. Melamud, Stanford University
Mirrored at CMU 16-311 Introduction to Robotics, http://generalrobotics.org

Forward Kinematics: Joint Angles to Bone Coordinates

- We will use the vector:

$$\mathbf{\Phi} = [\phi_1 \quad \phi_2 \quad \ldots \quad \phi_M]$$

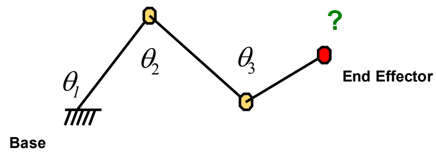
to represent the array of M joint DOF values
- We will also use the vector:

$$\mathbf{e} = [e_1 \quad e_2 \quad \ldots \quad e_N]$$

to represent an array of N DOFs that describe the end effector in world space. For example, if our end effector is a full joint with orientation, **e** would contain 6 DOFs: 3 translations and 3 rotations. If we were only concerned with the end effector position, **e** would just contain the 3 translations.

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005 – http://bit.ly/f0ViAN

Review [3]: Forward Kinematics

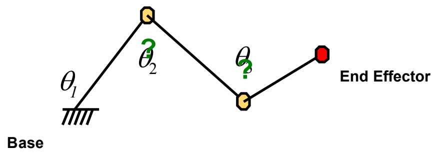


$\vec{\mathbf{x}} = \mathrm{f}(\vec{\boldsymbol{\theta}})$ (Choi) $\qquad \mathbf{e} = f(\mathbf{\Phi})$ (Rotenberg)

Adapted from slides ♥ 2002 K. J. Choi, Seoul National University
Graphics and Media Lab (http://graphics.snu.ac.kr) – mirrored at: http://bit.ly/hnzSAN

Review [4]: Inverse Kinematics

For more on characters & IK, see: Advanced Topics in CG Lecture 05



$\vec{\boldsymbol{\theta}} = \mathrm{f}^{-1}(\vec{\mathbf{x}})$ (Choi) $\qquad \mathbf{\Phi} = f^{-1}(\mathbf{e})$ (Rotenberg)

Adapted from slides ♥ 2002 K. J. Choi, Seoul National University
Graphics and Media Lab (http://graphics.snu.ac.kr) – mirrored at: http://bit.ly/hnzSAN

Inverse Kinematics: Issues

- IK is challenging because while f() may be relatively easy to evaluate, $f^{-1}()$ usually isn't
- For one thing, there may be several possible solutions for $\mathbf{\Phi}$, or there may be no solutions
- Even if there is a solution, it may require complex and expensive computations to find it
- As a result, there are many different approaches to solving IK problems

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005 – http://bit.ly/f0ViAN

Review [5]: Inverse Kinematics Demos

Inverse Kinematics demo

© 2008 M. Kinzelman
http://youtu.be/f52yZ491kPo

Inverse Kinematics Demonstration in Maya

© 2007 A. Brown
http://youtu.be/6JdLOLazJJ0

Momentum-based Inverse Kinematics with Motion Capture

© 2008 T. Komura, H. S. Lim, & R. W. H. Lau
http://youtu.be/FJTBMnP6oCM

PUMA robot playing golf

© 2011 K. Iyer
http://youtu.be/YvRBWIRAPsE

Review [6]: Ragdoll Physics

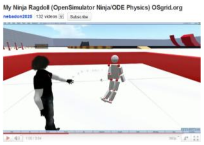
- Type of Procedural Animation
  - Automatically generates CGA directives (rotations)
  - Based on simulation
  - Rigid-body dynamics
- Articulated Figure
  - Gravity
  - No autonomous movement
  - Used for inert body
    - Usually: character death (car impact, falling body, *etc.*)
    - Less often: unconscious, paralyzed character
- Collisions with Multiple Bodies
  - Inter-character
  - Character-object

Falling Bodies © 1997 – 2001 Animats
http://www.animats.com

Review [7]: Ragdoll Physics Demos

© 2007 N. Picouet
http://youtu.be/ohNqCb--aSs

© 2006 P. Pelt
http://youtu.be/6JdLOLazJJ0
See also: http://youtu.be/5_QlsI0fyaU



© 2009 M. E. Cerquoni
http://youtu.be/uW_DK2qvKv8

© 2010 M. Heinzen (Arkaein)
http://bit.ly/gUj9Su / http://youtu.be/FJTBMnP6oCM

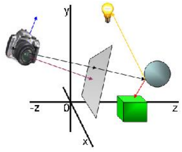
Review [8]: Physically-Based Modeling (PBM)

- Particle Dynamics
  - Emitters
    - 0-D (points), 1-D (lines), 2-D (planes, discs, cross-sections)
    - *e.g.*, fireworks (0-D); fountains (0/1/2-D); smokestacks, jets (2-D)
  - Simulation: birth-death process, functions of particle age/trajectory
- Rigid-Body Dynamics
  - Constrained systems of connected parts
  - Examples: falling rocks, colliding vehicles, rag dolls
- Articulated Figures: Primarily IK
- More References
  - ACM, *Intro to Physically-Based Modeling*: http://bit.ly/hhQvXd
  - Wikipedia, *Physics Engine*: http://bit.ly/h4PIRt
  - Wikipedia, *N-Body Problem*: http://bit.ly/1ayWwe
- PBM System: nVidia (Ageia) *PhysX*: http://bit.ly/cp7bnA

Rocks fall
Everyone dies

Ray Tracing [1]: Overview

- What is it?
- Why use it?
- Basics
- Advanced topics
- References



© 2006 – 2007 H. Kuijpers, Capgemini Netherlands
http://bit.ly/erkKrC

Adapted from slides ♥ 2001 D. Shreiner & B. Grantham, SCU
COEN 290: Computer Graphics I, Winter 2001 – http://bit.ly/hz1kfU

Ray Tracing [2]: Why Use It?

- 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 ♥ 2001 D. Shreiner & B. Grantham, SCU
COEN 290: Computer Graphics I, Winter 2001 – http://bit.ly/hz1kfU

Ray Tracing [3]: Who Uses It?

- Entertainment (Movies, Commercials)
- Games pre-production
- Simulation

Adapted from slides ♥ 2001 D. Shreiner & B. Grantham, SCU
COEN 290: Computer Graphics I, Winter 2001 – http://bit.ly/hz1kfU

Ray Tracing [4]: Brief History

- Decartes, 1637 A.D. - analysis of rainbow
- Arthur Appel, 1968 - used for lighting 3D models
- Turner Whitted, 1980 - “An Improved Illumination Model for Shaded Display” really kicked everyone off.
- 1980-now - Lots of research

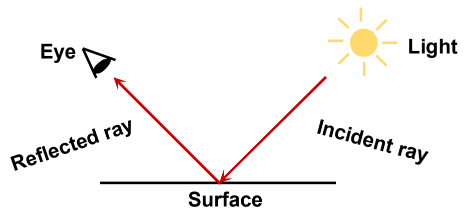
Adapted from slides ♥ 2001 D. Shreiner & B. Grantham, SCU
COEN 290: Computer Graphics I, Winter 2001 – http://bit.ly/hz1kfU

Ray Tracing [5]: Basic Operations

- Generating Rays
- Intersecting Rays with Scene
- Lighting
- Shadowing
- Reflections

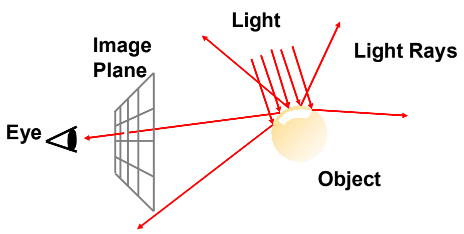
Ray Tracing [6]: Basic Idea

- Simulate light rays from light source to eye



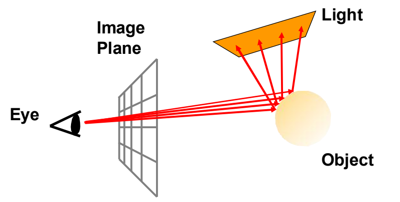
"Forward" Ray Tracing

- Trace rays from light
- Lots of work for little return



"Backward" Ray Tracing

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

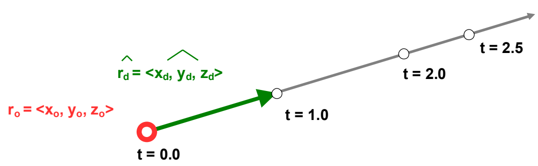


This is what most people mean by "ray tracing".

Ray: Parametric Form

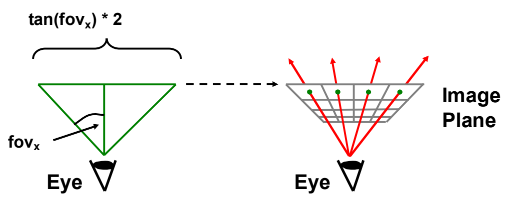
- Ray expressed as function of a single parameter ("t")

$$\langle x, y, z\rangle = \langle x_o, y_o, z_o\rangle + t * \widehat{\langle x_d, y_d, z_d\rangle}$$
$$\langle x, y, z\rangle = r_o + t\hat{r}_d$$



Generating Rays [1]

- Trace one ray for each pixel (*u*, *v*) in image plane

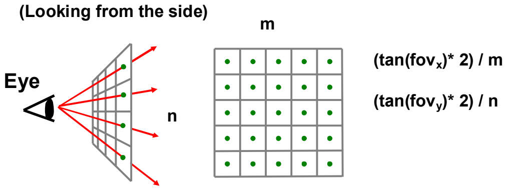


(Looking down from the top)

Generating Rays [2]

- Trace one ray for each pixel (u, v) in image plane

(Looking from the side)



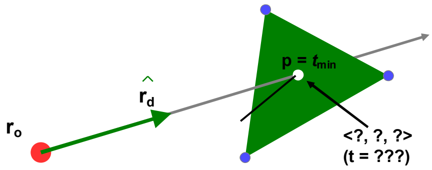
Generating Rays [3]

- Trace one ray for each pixel (u, v) in image plane

```
renderImage(){
    for each pixel i, j in the image
        ray.setStart(0, 0, 0);    // ro
        ray.setDir  ((.5 + i) * tan(fovx)* 2 / m,
                     (.5 + j) * tan(fovy)* 2 / n,
                     1.0);             // rd
        ray.normalize();
        image[i][j] = rayTrace(ray);
}
```

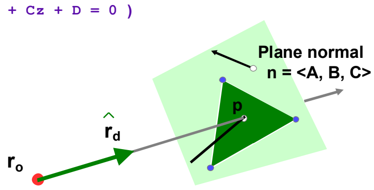

Ray/Triangle Intersection [1]: Intersection Test Revisited

- Want to know: at what *point* p does ray intersect triangle?
- Compute lighting, reflected rays, shadowing *from that point*



Ray/Triangle Intersection [2]: Ray/Plane Intersection Point *p*

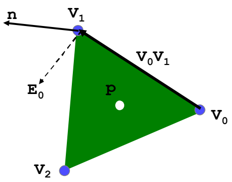
- Step 1 : Intersect with plane
  (Ax + By + Cz + D = 0)



t_{min} = p = -(n $\cdot$ r_o + D) / (n • r_d)

Ray/Triangle Intersection [3]: Triangle Containment

- Step 2 : Check against triangle edges

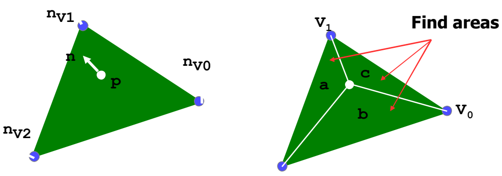


$E_i = V_iV_{i+1} \times n$ (plane A, B, C)
$d_i = -A \cdot N$ (plane D)

Plug p into (p • E_i + d_i) for each edge if signs are all positive or negative, point is inside triangle!

Triangle Normals for Shading

- Could use plane normals (flat shading)
- Better to interpolate from vertices



31

Finding Intersections

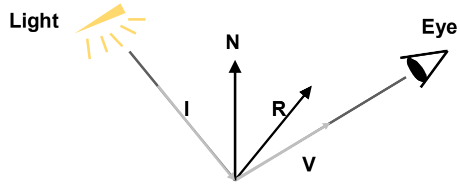
- Check all triangles, keep closest intersection t_{min}

```
hitObject(ray) {
   for each triangle in scene
      does ray intersect triangle?
      if(intersected and was closer)
         save that intersection
   if(intersected)
      return intersection point and normal
}
```


32

Lighting [1]: General Notation Review

- We'll use triangles for lights
- Can build complex shapes from triangles
- Some lighting terms



33

Lighting [2]: Modified Phong Illumination

- Use modified Phong lighting
  - similar to OpenGL
  - simulates rough and shiny surfaces

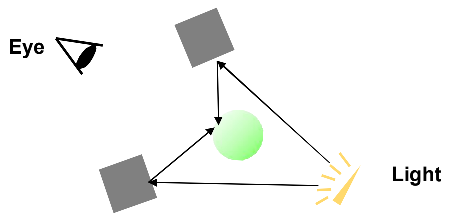
for each light

$$I_n = I_{ambient}K_{ambient} + I_{diffuse}K_{diffuse}\ (L\cdot N) + I_{specular}K_{specular}\ (R\cdot V)^n$$

34

Ambient Light

- $I_{ambient}$ – simulates indirect lighting in a scene

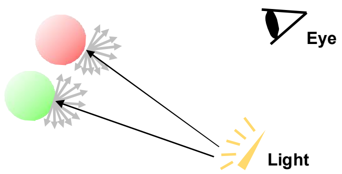


- May not need for RT!

35

Diffuse Light

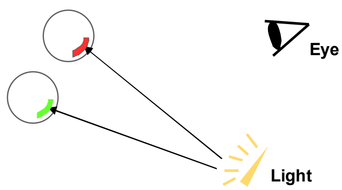
- $I_{diffuse}$ – simulates direct lighting on rough surface
- Viewer-independent
- Paper, rough wood, brick, etc...



36

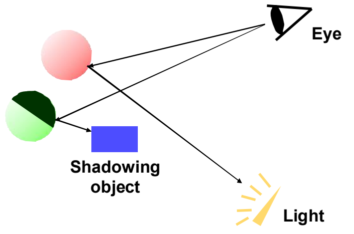
Specular Light

- $I_{specular}$ simulates direct lighting on a smooth surface
- Viewer dependent
- Plastic, metal, polished wood, etc...



Shadow Test

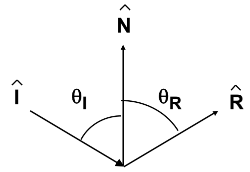
- Check against other objects to see if point is shadowed



Reflection

- Angle of incidence = angle of reflection ($\theta_I = \theta_R$)
- I, R, N lie in the same plane

$R = I - 2 (N \bullet I) N$

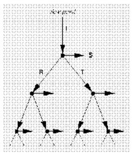


Putting It All Together [1]: Recursive Calculation & Ray Tree

- Recursive ray evaluation

```
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



Ray tree
© 2000 N. Patrikalakis, MIT
http://bit.ly/fjcGGk

I = Incident ray
S = light Source vector (*aka* L)
R = reflected ray
T = transmitted ray

Putting It All Together [2]: Applying Lighting

- Calculating surface color

```
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);
}
```


Putting It All Together [3]: Main Program

```
main() {
    triangles = readTriangles();
    image = renderImage(triangles);
    writeImage(image);
}
```


Good Start: What next?

- Lighting, Shadows, Reflection are enough to make some compelling images
- Want better lighting and objects
- Need more speed

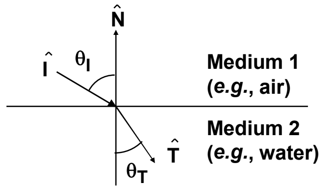
More Quality, More Speed

- Better Lighting + Forward Tracing
- Texture Mapping
- Modeling Techniques
- Distributed Ray Tracing: Techniques
  - Motion Blur
  - Depth of Field
  - Blurry Reflection/Refraction
  - Wikipedia, *Distributed Ray Tracing*: http://bit.ly/ihyVUs
- Improving Image Quality
- Acceleration Techniques

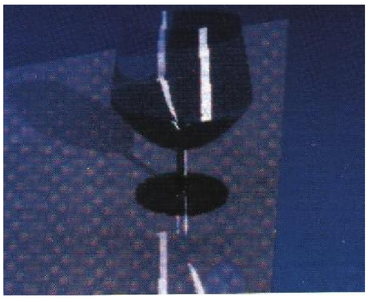
Refraction [1]: Snell's Law

- Keep track of medium (air, glass, etc)
- Need *index of refraction* (η)
- Need solid objects

$$\frac{\sin(\theta_I)}{\sin(\theta_T)} = \frac{\eta_1}{\eta_2}$$



Refraction [2]: Example



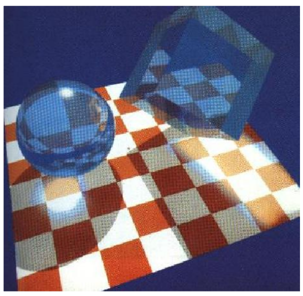
Improved Light Model: Cook-Torrance

- Cook-Torrance model
  - Based on a microfacet model
  - Wikipedia: http://bit.ly/hX3U30
- Metals have different color at angle
- Oblique reflections leak around corners

Using "Forward" Ray Tracing [1]: Lensed Caustics for Indirect Lighting

- Backward tracing doesn't handle indirect lighting too well
- To get *caustics*, trace forward, store results in texture map

Using "Forward" Ray Tracing [2]: Example



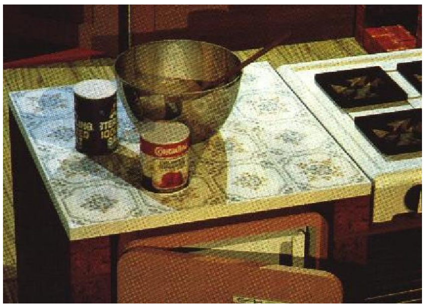
49

Texture Mapping & Ray Tracing [1]: Applying Surface Detail

- Using texture maps
  - Add surface detail
  - Think of it like texturing in OpenGL
- Diffuse, specular colors
- Shininess value
- Bump map
- Transparency value

50

Texture Mapping & Ray Tracing [2]: Example



51

Parametric Surfaces

- More expressive than triangle
- Intersection is probably slower
- *u* and *v* on surface can be used as texture *s*, *t*

52

Constructive Solid Geometry

- Union, Subtraction, Intersection of solid objects
- Have to keep track of intersections

53

Hierarchical Transformation

- Scene made of parts
- Each part made of smaller parts
- Each smaller part has transformation linking it to larger part
- Transformation can change over time: animation (CGA)

54

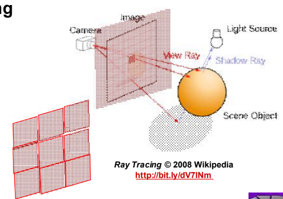
Distributed Ray Tracing [1]: Basic Idea

- Average multiple rays instead of just one ray
- Use for both shadows, reflections, transmission (refraction)
- Use for motion blur
- Use for depth of field

Distributed Ray Tracing [2]: Example

Distributed Ray Tracing [3]: Supersampling

- 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



Ray Tracing © 2008 Wikipedia
http://bit.ly/dV7lNm

Acceleration!

- 1280x1024 image with 10 rays/pixel
- 1000 objects (triangle, CSG, NURBS)
- 3 levels recursion

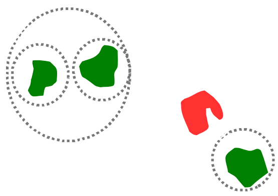
39321600000 intersection tests

100000 tests/second -> 109 days!

Must use an acceleration method!

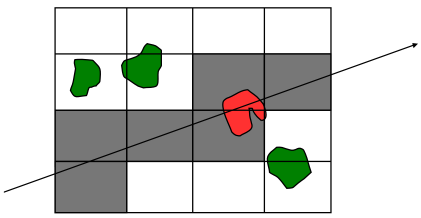
Bounding Volumes

- Use simple shape for quick test, keep BV hierarchy



Spatial Partitioning: Subdivision

- Break your space into pieces
- Search the structure linearly



Parallelism

- Can always throw more processors at it
- Parallel computing model
  - Multiple processes or threads
  - Data parallel: separate pixel for each

61

Really Advanced Stuff

- Error analysis
- Hybrid radiosity/ray-tracing
- Metropolis Light Transport
- Memory-Coherent Ray-tracing

Adapted from slides ♥ 2001 D. Shreiner & B. Grantham, SCU
COEN 290: Computer Graphics I, Winter 2001 – http://bit.ly/hz1kfU

62

References

- *Introduction to Ray-Tracing*, Glassner *et al.*, 1989, 0-12-286160-4
- *Advanced Animation and Rendering Techniques*, Watt & Watt, 1992, 0-201-54412-1
- *Computer Graphics: Image Synthesis*, Joy *et al.*, 1988, 0-8186-8854-4
- SIGGRAPH Proceedings (All)

Adapted from slides ♥ 2001 D. Shreiner & B. Grantham, SCU
COEN 290: Computer Graphics I, Winter 2001 – http://bit.ly/hz1kfU

63

Summary

- Reading for Last Class: §5.3, Eberly 2^e; CGA Handout
- Reading for Today: Chapter 14, Eberly 2^e
- Reading for Next Class: Ray Tracing Handout
- Last Class: Particle Systems, Collisions, IK/CGA Concluded
  - Dynamics *vs.* kinematics, forward *vs.* inverse revisited
  - IK: autonomous *vs.* hand-animated; solution approaches
  - Rag doll physics, rigid-body dynamics, physically-based models
- Today: Ray Tracing, Part 1 of 2
  - Vectors
    - Light (L): to point light sources (or shadows)
    - Reflected (R): from object surface
    - Transmitted or Transparency (T): through transparent object
  - t_{min}: distance to intersection between ray and bounding volume
  - Ways to find t_{min}
  - Basic recursive ray tracing: ray trees

64

Terminology

- Joints: Parts of Robot / Articulated Figure That Turn, Slide
- Effectors: Parts of Robot / Articulated Figure That Act (*e.g.*, Hand, Foot)
- Bones: Effectors, Other Parts That Rotate about, Slide through Joints
- Procedural Animation: Automatic Generation of Motion *via* Simulation
- 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
  - Calculate vectors emanating from world-space coordinate of t_{min}
    - Light (L): 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 found
    - Builds ray tree rooted at intersection point
    - Base cases: unobstructed vector to light; depth limit
  - Parallel RT: use multiple threads/processes for each (u, v) or t