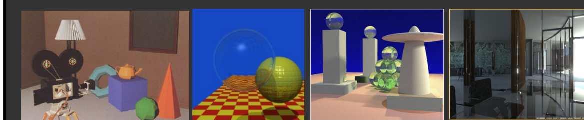
Computer Graphics II: Rendering

CSE 168 [Spr 25], Lecture 1: Overview and Ray Tracing

Ravi Ramamoorthi

http://viscomp.ucsd.edu/classes/cse168/sp25/



1

Goals

- **Systems:** Write a modern 3D image synthesis program (path tracer with importance sampling)
- **Theory:** Mathematical aspects and algorithms underlying modern physically-based rendering
- **Topics:** Other modern topics like image-based, real-time, precomputed, volumetric rendering
- This course is ***not*** about the specifics of 3D rendering software like PBRT, Mitsuba etc. New, we optionally encourage OptiX, a real-time raytracing API for NVIDIA GPUs

2

Instructor

Ravi Ramamoorthi http://www.cs.ucsd.edu/~ravir

- PhD Stanford, 2002 [with Pat Hanrahan, 2020 Turing Award] "*Spherical Harmonic Lighting*" widely used in games (e.g. Halo series), movies (e.g. Avatar), etc. (Adobe, …)
- At Columbia 2002-2008, UC Berkeley 2009-2014
- *"Monte Carlo denoising"* inspired raytracing offline, real-time
- At UCSD since Jul 2014: Director, Center for Visual Computing
- Awards for research: White House PECASE (2008), SIGGRAPH Significant New Researcher (2007), ACM Fellow
- https://www.youtube.com/watch?v=qpyCXqXGe7I
- Computer Graphics online MOOC (CSE 167x) finalist for two edX Prizes. Will use CSE 168 MOOC on edX edge as a feedback system

3

Course Staff

- Ravi Ramamoorthi, ravir@cs.ucsd.edu
- Teaching Assistants:
  - Nithin Raghavan [n2raghavan@ucsd.edu]
  - Please see website (and piazza) for office hours etc.

4

Rendering: 1960s (visibility)

- Roberts (1963), Appel (1967) - hidden-line algorithms
- Warnock (1969), Watkins (1970) - hidden-surface
- Sutherland (1974) - visibility = sorting

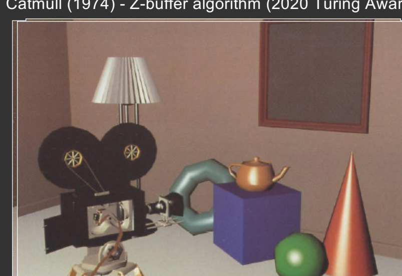
Images from FvDFH, Pixar's Shutterbug
Slide ideas for history of Rendering courtesy Marc Levoy

5

Rendering: 1970s (lighting)

1970s - raster graphics

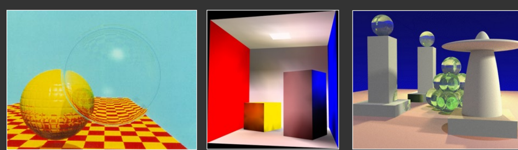
- Gouraud (1971) - diffuse lighting, Phong (1974) - specular lighting
- Blinn (1974) - curved surfaces, texture
- Catmull (1974) - Z-buffer algorithm (2020 Turing Award)



6

Rendering (1980s, 90s: Global Illumination)

early 1980s - global illumination
- Whitted (1980) - ray tracing
- Goral, Torrance et al. (1984) radiosity
- Kajiya (1986) - the rendering equation, path tracing (this is what this course is about, modern rendering)



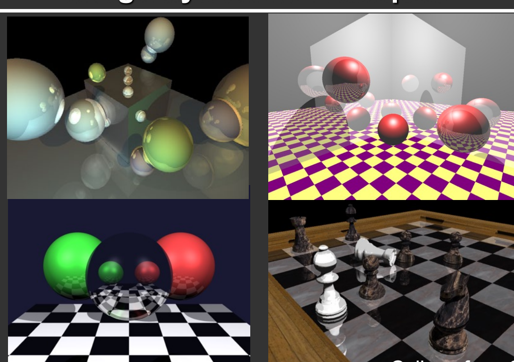
7

Why Study Computer Graphics Rendering?

- Applications (Movies, Games, Digital Advertising, Lighting Simulation, Digital Humans, Virtual Reality)
- Fundamental Intellectual Challenges
  - Create *photorealistic* virtual world
  - Understand physics *and* computation of light transport
  - Physically-based rendering has replaced ad-hoc approaches in industry (offline ~ 2011, real-time ~2018)
- Beautiful Imagery: Realistic Computer Graphics
  - 2020 Turing Award given for CGI in Filmmaking
- Assume taken CSE 167 or equivalent (+done well)
  - This is a challenging course, work starts immediately
  - (First 2 weeks on raytracing may be review for some)

8

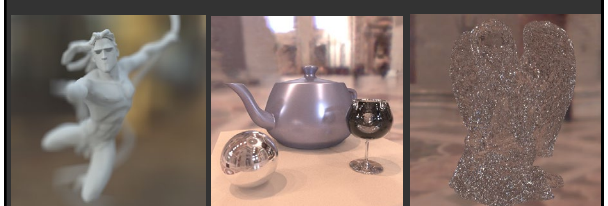
Image Synthesis Examples



Collage from 2007

9

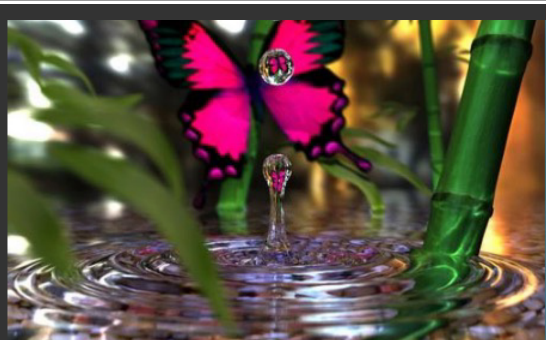
From UCB CS 294 16 years ago



Daniel Ritchie and Lita Cho

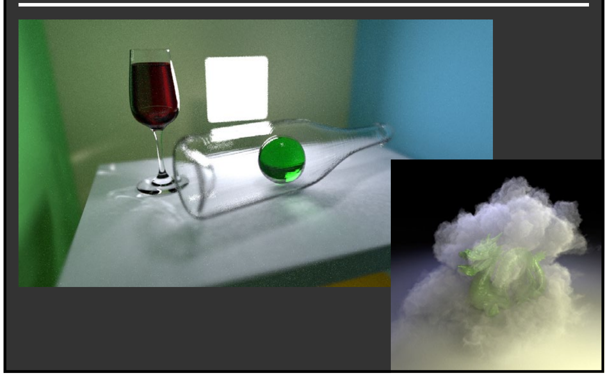
10

CSE 168 Contest 2007: Butterfly



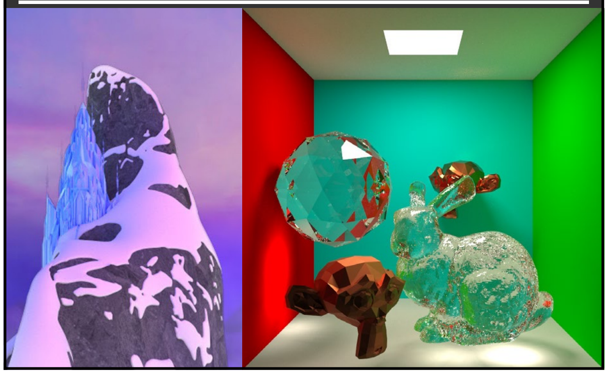
11

CSE 168 Spring 2020



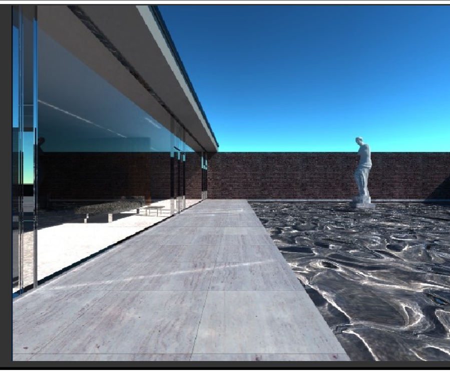
12

CSE 168 Spring 2021



13

Mies House: Swimming Pool



14

Logistics

- Website https://viscomp.ucsd.edu/classes/cse168/sp25/168.html has most of the information (look at it carefully)
- *We will be leveraging full MOOC infrastructure (use private SPOC)*
  - Please join course course on edX edge: **DEMO**
  - Compulsory for most assignments, feedback systems
  - *Must still submit "official" CSE 168 assignment (see website)*
  - Please do ask us if you are confused; we are here to help
  - No required texts; optional PBRT book, Digital Image Synthesis
  - Office hours: after class (Tu/Thu 11-12) in CSE 4118
- Course newsgroup on Piazza, or can use edX edge directly
- Website for late, collaboration policy (groups of 2), etc
- Do try to attend class sessions (and discussions, keep assigned section)
- Questions?

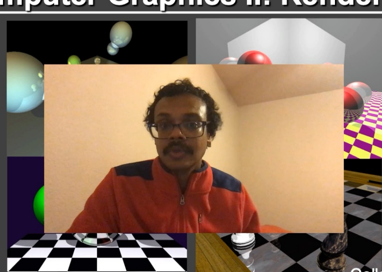
15

This is a Modernized Course

- Teach Modern Physically-Based Rendering and Path Tracing, as used in industry (Prof. consulted with Pixar on change to physically-based shading, importance sampling in 2011, written many key papers; consults NVIDIA)
- Emphasis on step-by-step development, get it right (lots of subtle math, compare to reference solutions)
- Focus on offline but discuss real-time, image-based, PRT
- Homework starts right away, due in 2 weeks
- New developments: NVIDIA OptiX ray-tracing API like OpenGL, since 2018 RTX cards 10G rays/second Video
- Encourage (but optional) use of OptiX. If you use this, setup yourself but basic skeleton provided. Or really slow.

16

Computer Graphics II: Rendering



Collage from 2007

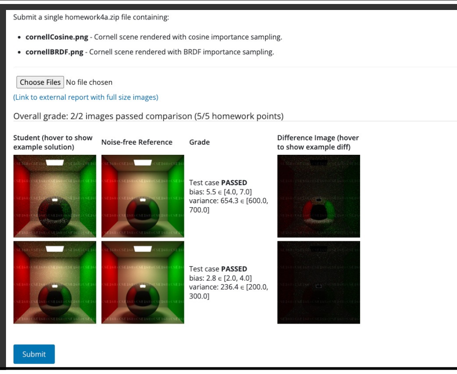
17

Innovation: Feedback Servers

- Feedback/Grading servers for homeworks 1-4
- Submit images, compare to original
  - Program generates difference images, report url
  - Can get feedback multiple times; submit final url
  - All run on edX edge (also have alternative standalone)
- "Feedback" not necessarily grading
  - Can run extra test cases, look at code, grade fairly
  - But use of feedback servers/edX edge is mandatory
  - Note for this course; unlike 167 results not deterministic, will give information re noise/variance
  - Can use any laptop/desktop, do it offline or in OptiX
- Will test out with HW 1 images

18

Demo of edX edge, Feedbacks



19

Workload

- Lots of fun, rewarding but may involve significant work
  - Previous reviews: "Undergraduate in name only" "Most time consuming course at UCSD"
- 5 programming projects; almost all are time-consuming. Can be done in groups of two. START EARLY !!
- Graded entirely on programming, weights on 168 website
  - Writeups on 168 website for assignments really good, look at them
- Prerequisites: CSE 167, did well, enjoyed it
- First homework last assignment in my CSE 167
  - Little bit of sink or swim to continue in course (but we will also provide OptiX, embree references after assignment is due)
  - But not everyone has done a raytracer before, some additional requirements for those who have already done one
- Should be a difficult, but fun and rewarding course

20

Quick Inclusion Note

Since I do occasionally get asked this question:

- You are welcome to take this course if color-blind
  - Let me know if I create too many red-green metamers
  - Some of the best-known computer graphics researchers have been color-blind (ask re some stories)
- And for most other vision issues
  - We've even had computer graphics award winners who have been extremely nearsighted (legally blind)

21

CSE 168 is only a first step

- *If you enjoy CSE 168 and do well*:
- CSE 165 (VR course; Schulze)
- Next winter: 169 (Animation)
- Graduate: CSE 272, 274 (Topics), 273, many 291s

22

To Do

- Look at website
- Various policies for course. E-mail if confused.
- Sign up for edX edge, Piazza, etc.
- Skim assignments if you want. All are ready
- Assignment 1, Due Apr 14 (see website).
- Any questions?
- Start now with raytracing lecture

23

Effects needed for Realism

- (Soft) Shadows
- Reflections (Mirrors and Glossy)
- Transparency (Water, Glass)
- Interreflections (Color Bleeding)
- Complex Illumination (Natural, Area Light)
- Realistic Materials (Velvet, Paints, Glass)
- And many more

24

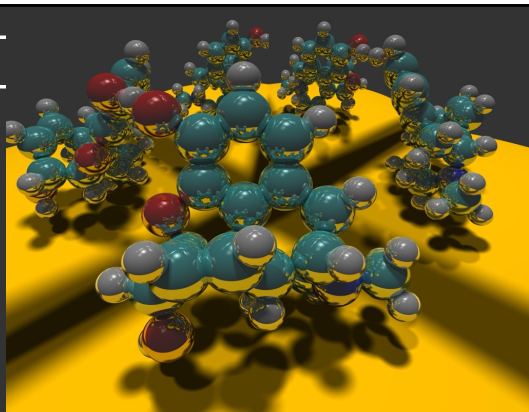


Image courtesy Paul Heckbert 1983

25

Ray Tracing

- Different Approach to Image Synthesis as compared to Hardware pipeline (OpenGL)
- Pixel by Pixel instead of Object by Object
- Easy to compute shadows/transparency/etc

26

Outline

- *History*
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Shadows / Reflections (core algorithm)
- Optimizations
- Current Research

27

Ray Tracing: History

- Appel 68
- Whitted 80 [recursive ray tracing]
  - Landmark in computer graphics
- Lots of work on various geometric primitives
- Lots of work on accelerations
- Current Research
  - Real-Time raytracing (historically, slow technique)
  - Ray tracing architecture

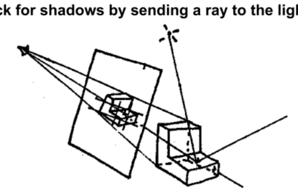
28

Ray Tracing History

Ray Tracing in Computer Graphics

Appel 1968 - Ray casting

1. Generate an image by sending one ray per pixel
2. Check for shadows by sending a ray to the light



CS348B Lecture 2 Pat Hanrahan, Spring 2009

29

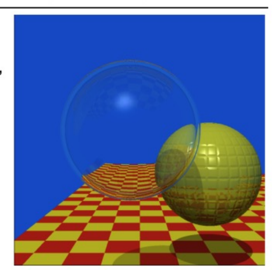
Ray Tracing History

Ray Tracing in Computer Graphics

"An improved Illumination model for shaded display," T. Whitted, CACM 1980

Resolution:
512 x 512

Time:
VAX 11/780 (1979)
74 min.
PC (2006)
6 sec.



Spheres and Checkerboard, T. Whitted, 1979

CS348B Lecture 2 Pat Hanrahan, Spring 2009

30

From SIGGRAPH 18

Real Photo: Instructor and Turner Whitted at SIGGRAPH 18

31

Outline

- History
- *Basic Ray Casting (instead of rasterization)*
  - Comparison to hardware scan conversion
- Shadows / Reflections (core algorithm)
- Optimizations
- Current Research

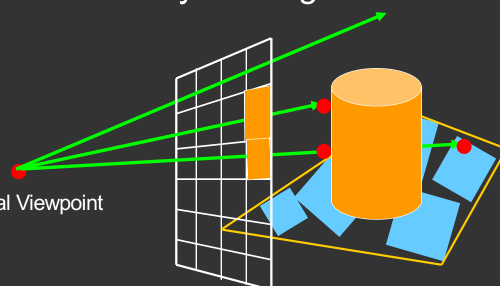
32

Ray Casting

Produce same images as with OpenGL
- Visibility per pixel instead of Z-buffer
- Find nearest object by shooting rays into scene
- Shade it as in standard OpenGL

33

Ray Casting



34

Comparison to hardware scan-line

- Per-pixel evaluation, per-pixel rays (not scan-convert each object). On face of it, costly
- But good for walkthroughs of extremely large models (amortize preprocessing, low complexity)
- More complex shading, lighting effects possible

35

Outline in Code

```
Image Raytrace (Camera cam, Scene scene, int width, int height)
{
   Image image = new Image (width, height) ;
   for (int i = 0 ; i < height ; i++)
        for (int j = 0 ; j < width ; j++) {
                Ray ray = RayThruPixel (cam, i, j) ;
                Intersection hit = Intersect (ray, scene) ;
                image[i][j] = FindColor (hit) ;
                }
   return image ;
}
```

36

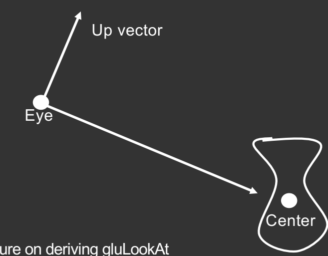
Finding Ray Direction

- Goal is to find ray direction for given pixel i and j
- Many ways to approach problem
  - Objects in world coord, find dirn of each ray (we do this)
  - Camera in canonical frame, transform objects (OpenGL)
- Basic idea
  - Ray has origin (camera center) and direction
  - Find direction given camera params and i and j
- Camera params as in gluLookAt
  - Lookfrom[3], LookAt[3], up[3], fov

37

Similar to gluLookAt derivation

- gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)
- Camera at eye, looking at center, with up direction being up



From 167 lecture on deriving gluLookAt

38

Constructing a coordinate frame?

We want to associate **w** with **a**, and **v** with **b**

- But **a** and **b** are neither orthogonal nor unit norm
- And we also need to find **u**

$$w = \frac{a}{\|a\|}$$

$$u = \frac{b \times w}{\|b \times w\|}$$

$$v = w \times u$$

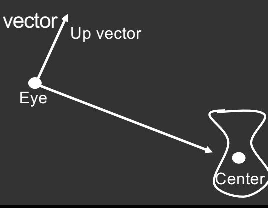
From 167 basic math lecture - Vectors: Orthonormal Basis Frames

39

Camera coordinate frame

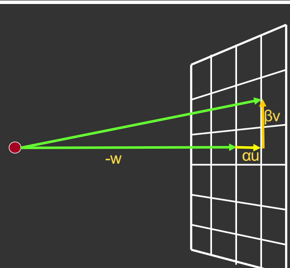
$$w = \frac{a}{\|a\|} \qquad u = \frac{b \times w}{\|b \times w\|} \qquad v = w \times u$$

- We want to position camera at origin, looking down –Z dirn
- Hence, vector **a** is given by **eye** – **center**
- The vector **b** is simply the **up** vector



40

Canonical viewing geometry



$$ray = eye + t\frac{\alpha u + \beta v - w}{|\alpha u + \beta v - w|}$$

$$\alpha = \tan\left(\frac{fovx}{2}\right) \times \left(\frac{j - (width/2)}{width/2}\right) \qquad \beta = \tan\left(\frac{fovy}{2}\right) \times \left(\frac{(height/2) - i}{height/2}\right)$$

41

Outline in Code

```
Image Raytrace (Camera cam, Scene scene, int width, int height)
{
   Image image = new Image (width, height) ;
   for (int i = 0 ; i < height ; i++)
      for (int j = 0 ; j < width ; j++) {
            Ray ray = RayThruPixel (cam, i, j) ;
            Intersection hit = Intersect (ray, scene) ;
            image[i][j] = FindColor (hit) ;
            }
   return image ;
}
```

42

Ray/Object Intersections

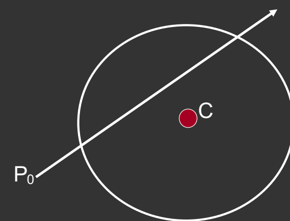
- Heart of Ray Tracer
  - One of the main initial research areas
  - Optimized routines for wide variety of primitives
- Various types of info
  - Shadow rays: Intersection/No Intersection
  - Primary rays: Point of intersection, material, normals
  - Texture coordinates
- Work out examples
  - Triangle, sphere, polygon, general implicit surface

43

Ray-Sphere Intersection

$$ray \equiv \vec{P} = \vec{P}_0 + \vec{P}_1 t$$

$$sphere \equiv (\vec{P} - \vec{C}) \bullet (\vec{P} - \vec{C}) - r^2 = 0$$



44

Ray-Sphere Intersection

$$ray \equiv \vec{P} = \vec{P}_0 + \vec{P}_1 t$$

$$sphere \equiv (\vec{P} - \vec{C}) \bullet (\vec{P} - \vec{C}) - r^2 = 0$$

Substitute

$$ray \equiv \vec{P} = \vec{P}_0 + \vec{P}_1 t$$

$$sphere \equiv (\vec{P}_0 + \vec{P}_1 t - \vec{C}) \bullet (\vec{P}_0 + \vec{P}_1 t - \vec{C}) - r^2 = 0$$

Simplify

$$t^2(\vec{P}_1 \bullet \vec{P}_1) + 2t\,\vec{P}_1 \bullet (\vec{P}_0 - \vec{C}) + (\vec{P}_0 - \vec{C}) \bullet (\vec{P}_0 - \vec{C}) - r^2 = 0$$

45

Ray-Sphere Intersection

$$t^2(\vec{P}_1 \bullet \vec{P}_1) + 2t\,\vec{P}_1 \bullet (\vec{P}_0 - \vec{C}) + (\vec{P}_0 - \vec{C}) \bullet (\vec{P}_0 - \vec{C}) - r^2 = 0$$

Solve quadratic equations for t

- 2 real positive roots: pick smaller root
- Both roots same: tangent to sphere
- One positive, one negative root: ray origin inside sphere (pick + root)
- Complex roots: no intersection (check discriminant of equation first)

46

Ray-Sphere Intersection

- Intersection point: $ray \equiv \vec{P} = \vec{P}_0 + \vec{P}_1 t$
- Normal (for sphere, this is same as coordinates in sphere frame of reference, useful other tasks)

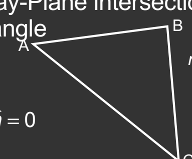
$$normal = \frac{\vec{P} - \vec{C}}{|\vec{P} - \vec{C}|}$$

47

Ray-Triangle Intersection

- One approach: Ray-Plane intersection, then check if inside triangle
- Plane equation:

$$plane \equiv \vec{P} \bullet \vec{n} - \vec{A} \bullet \vec{n} = 0$$

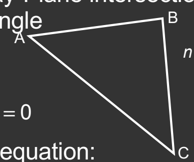


$$n = \frac{(C-A)\times(B-A)}{|(C-A)\times(B-A)|}$$

48

Ray-Triangle Intersection

- One approach: Ray-Plane intersection, then check if inside triangle
- Plane equation:



$$n = \frac{(C-A)\times(B-A)}{\left|(C-A)\times(B-A)\right|}$$

$$plane \equiv \vec{P}\cdot\vec{n} - \vec{A}\cdot\vec{n} = 0$$

- Combine with ray equation:

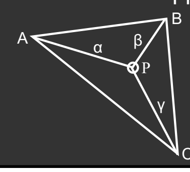
$$ray \equiv \vec{P} = \vec{P}_0 + \vec{P}_1 t$$

$$(\vec{P}_0 + \vec{P}_1 t)\cdot\vec{n} = \vec{A}\cdot\vec{n} \qquad t = \frac{\vec{A}\cdot\vec{n} - \vec{P}_0\cdot\vec{n}}{\vec{P}_1\cdot\vec{n}}$$

49

Ray inside Triangle

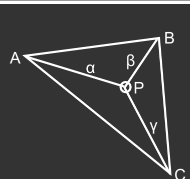
- Once intersect with plane, still need to find if in triangle
- Many possibilities for triangles, general polygons (point in polygon tests)
- We find parametrically [barycentric coordinates]. Also useful for other applications (texture mapping)



$$P = \alpha A + \beta B + \gamma C$$
$$\alpha \geq 0, \beta \geq 0, \gamma \geq 0$$
$$\alpha + \beta + \gamma = 1$$

50

Ray inside Triangle



$$P = \alpha A + \beta B + \gamma C$$
$$\alpha \geq 0, \beta \geq 0, \gamma \geq 0$$
$$\alpha + \beta + \gamma = 1$$

$$P - A = \beta(B-A) + \gamma(C-A)$$
$$0 \leq \beta \leq 1\ ,\ 0 \leq \gamma \leq 1$$
$$\beta + \gamma \leq 1$$

51

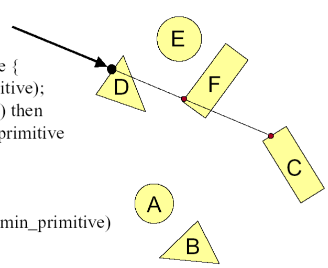
Other primitives

- Much early work in ray tracing focused on ray-primitive intersection tests
- Cones, cylinders, ellipsoids
- Boxes (especially useful for bounding boxes)
- General planar polygons
- Many more
- Consult chapter in Glassner (handed out) for more details and possible extra credit

52

Ray Scene Intersection

```
Intersection FindIntersection(Ray ray, Scene scene)
{
    min_t = infinity
    min_primitive = NULL
    For each primitive in scene {
        t = Intersect(ray, primitive);
        if (t > 0 && t < min_t) then
            min_primitive = primitive
            min_t = t
        }
    }
    return Intersection(min_t, min_primitive)
}
```



53

Transformed Objects

- E.g. transform sphere into ellipsoid
- Could develop routine to trace ellipsoid (compute parameters after transformation)
- May be useful for triangles, since triangle after transformation is still a triangle in any case
- But can also use original optimized routines

54

Ray-Tracing Transformed Objects

We have an optimized ray-sphere test
- But we want to ray trace an ellipsoid…

Solution: Ellipsoid transforms sphere
- Apply inverse transform to ray, use ray-sphere
- Allows for instancing (traffic jam of cars)
- Same idea for other primitives

55

Transformed Objects

- Consider a general 4x4 transform M
  - Will need to implement matrix stacks like in OpenGL
- Apply inverse transform M^{-1} to ray
  - Locations stored and transform in homogeneous coordinates
  - Vectors (ray directions) have homogeneous coordinate set to 0 [so there is no action because of translations]
- Do standard ray-surface intersection as modified
- Transform intersection back to actual coordinates
  - Intersection point p transforms as Mp
  - Distance to intersection if used may need recalculation
  - Normals n transform as $M^{-t}n$. Do all this before lighting

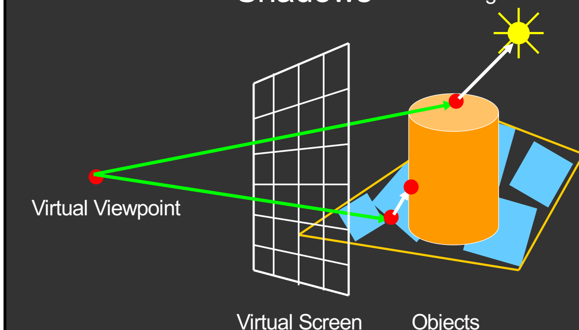
56

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- *Shadows / Reflections (core algorithm)*
- Optimizations
- Current Research

57

Shadows

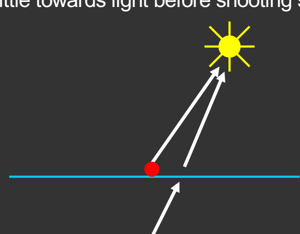


Shadow ray to light is blocked: object in shadow

58

Shadows: Numerical Issues

- Numerical inaccuracy may cause intersection to be below surface (effect exaggerated in figure)
- Causing surface to incorrectly shadow itself
- Move a little towards light before shooting shadow ray



59

Outline in Code

```
Image Raytrace (Camera cam, Scene scene, int width, int height)
{
  Image image = new Image (width, height) ;
  for (int i = 0 ; i < height ; i++)
     for (int j = 0 ; j < width ; j++) {
          Ray ray = RayThruPixel (cam, i, j) ;
          Intersection hit = Intersect (ray, scene) ;
          image[i][j] = FindColor (hit) ;
          }
  return image ;
}
```

60

Lighting Model

- Similar to OpenGL
- Lighting model parameters (global)
  - Ambient r g b
  - Attenuation const linear quadratic

$$L = \frac{L_0}{const + lin * d + quad * d^2}$$

- Per light model parameters
  - Directional light (direction, RGB parameters)
  - Point light (location, RGB parameters)

61

Material Model

- Diffuse reflectance (r g b)
- Specular reflectance (r g b)
- Shininess s
- Emission (r g b)
- All as in OpenGL

62

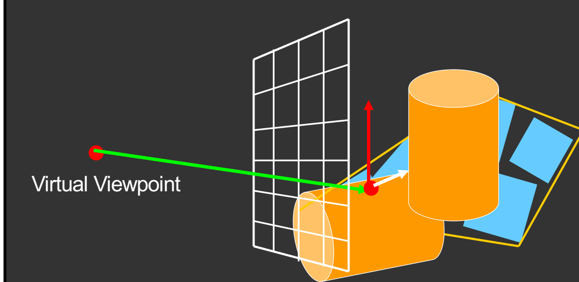
Shading Model

$$I = K_a + K_e + \sum_{i=1}^{n} V_i L_i (K_d \max(l_i \cdot n, 0) + K_s (\max(h_i \cdot n, 0))^s)$$

- Global ambient term, emission from material
- For each light, diffuse specular terms
- Note visibility/shadowing for each light (not in OpenGL)
- Evaluated per pixel per light (not per vertex)

63

Mirror Reflections/Refractions



Generate reflected ray in mirror direction,
Get reflections and refractions of objects

64

Recursive Ray Tracing

For each pixel
- Trace Primary Eye Ray, find intersection
- Trace Secondary Shadow Ray(s) to all light(s)
  - Color = Visible ? Illumination Model : 0 ;
- Trace Reflected Ray
  - Color += reflectivity * Color of reflected ray

65

Recursive Shading Model

$$I = K_a + K_e + \sum_{i=1}^{n} V_i L_i (K_d \max(l_i \cdot n, 0) + K_s (\max(h_i \cdot n, 0))^s) + K_s I_R + K_T I_T$$

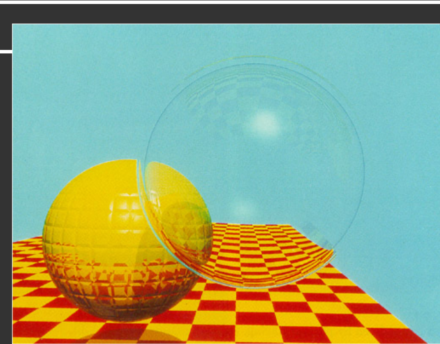
- Highlighted terms are recursive specularities [mirror reflections] and transmission (latter is extra credit)
- Trace secondary rays for mirror reflections and refractions, include contribution in lighting model
- GetColor calls RayTrace recursively (the I values in equation above of secondary rays are obtained by recursive calls)

66

Problems with Recursion

- Reflection rays may be traced forever
- Generally, set maximum recursion depth
- Same for transmitted rays (take refraction into account)

67



Turner Whitted 1980

68

Effects needed for Realism

- (Soft) Shadows
- Reflections (Mirrors and Glossy)
- Transparency (Water, Glass)
- Interreflections (Color Bleeding)
- Complex Illumination (Natural, Area Light)
- Realistic Materials (Velvet, Paints, Glass)

Discussed in this lecture
Not discussed but possible with distribution ray tracing
Hard (but not impossible) with ray tracing; radiosity methods
All are possible with path tracing developed in this course

69

Some basic add ons

- Area light sources and soft shadows: break into grid of n x n point lights
  - Use jittering: Randomize direction of shadow ray within small box for given light source direction
  - Jittering also useful for antialiasing shadows when shooting primary rays
- More complex reflectance models
  - Simply update shading model
  - But at present, we can handle only mirror global illumination calculations
- Some of these required for those who have already done a raytracer (167 with Chern or me)

70

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Shadows / Reflections (core algorithm)
- *Optimizations*
- Current Research

71

Acceleration

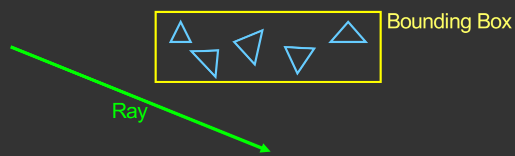
Testing each object for each ray is slow
- Fewer Rays
  Adaptive sampling, depth control
- Generalized Rays
  Beam tracing, cone tracing, pencil tracing etc.
- Faster Intersections
  - Optimized Ray-Object Intersections
  - ***Fewer Intersections***

72

Acceleration Structures

Bounding boxes (possibly hierarchical)

If no intersection bounding box, needn't check objects



Spatial Hierarchies (Oct-trees, kd trees, BSP trees)

73

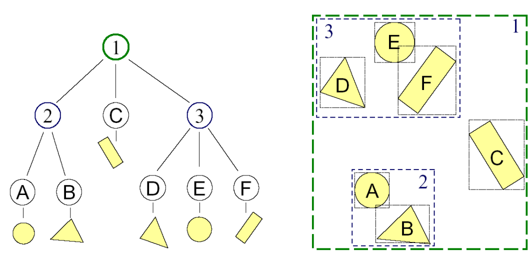
Ray Tracing Acceleration Structures

- Bounding Volume Hierarchies (BVH)
- Uniform Spatial Subdivision (Grids)
- Binary Space Partitioning (BSP Trees)
  - Axis-aligned often for ray tracing: kd-trees
- Conceptually simple, implementation a bit tricky
  - Lecture relatively high level: Start early
  - Remember that acceleration a small part of grade
  - But will struggle in future if developing in software

74

Bounding Volume Hierarchies 1

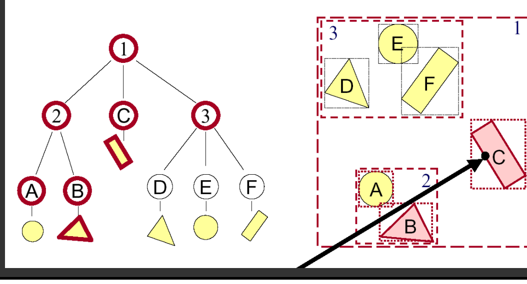
- Build hierarchy of bounding volumes
  - Bounding volume of interior node contains all children



75

Bounding Volume Hierarchies 2

- Use hierarchy to accelerate ray intersections
  - Intersect node contents only if hit bounding volume



76

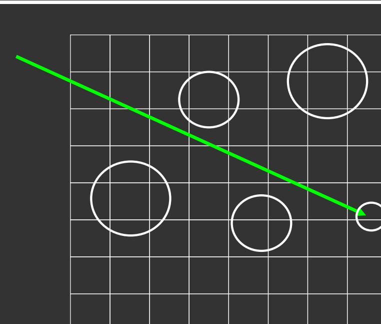
Bounding Volume Hierarchies 3

- Sort hits & detect early termination

```
FindIntersection(Ray ray, Node node)
{
    // Find intersections with child node bounding volumes
    ...
    // Sort intersections front to back
    ...
    // Process intersections (checking for early termination)
    min_t = infinity;
    for each intersected child i {
        if (min_t < bv_t[i]) break;
        shape_t = FindIntersection(ray, child);
        if (shape_t < min_t) { min_t = shape_t;}
    }
    return min_t;
}
```

77

Acceleration Structures: Grids



78

Acceleration and Regular Grids

- Simplest acceleration, for example 5x5x5 grid
- For each grid cell, store overlapping triangles
- March ray along grid (need to be careful with this), test against each triangle in grid cell
- More sophisticated: kd-tree, oct-tree bsp-tree
- Or use (hierarchical) bounding boxes
- Try to implement some acceleration in HW

79

Note on Optix, Code Reuse

- *No Copying Code previous students, solutions, or any online resources*
- *AI agents like chatGPT may be used only as a search engine for example for explaining the way specific commands are used, but you should not copy code from them.*
- *No posting code online including to github*
- Some students felt skeleton only for OptiX unfair
  - And in spring 20 tried copying to compensate. *Bad!!*
- Optix skeleton only Optix setup, no raytracing
  - Because writing from scratch in new language is hard
  - Acceleration structures are built-in, can use
  - *Still likely harder option, because of learning curve* (but great performance for course)

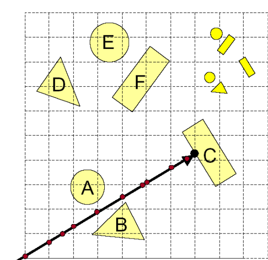
80

Uniform Grid: Problems

- Potential problem:
  - How choose suitable grid resolution?

Too little benefit if grid is too coarse

Too much cost if grid is too fine

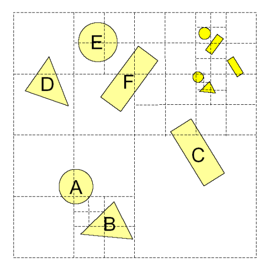


81

Octree

- Construct adaptive grid over scene
  - Recursively subdivide box-shaped cells into 8 octants
  - Index primitives by overlaps with cells

Generally fewer cells

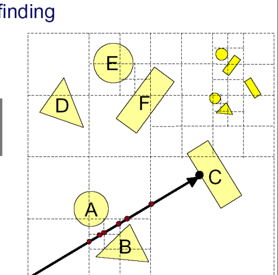


82

Octree traversal

- Trace rays through neighbor cells
  - Fewer cells
  - More complex neighbor finding

Trade-off fewer cells for more expensive traversal



83

Math of 2D Bounding Box Test

- Can you find a t in range

$t > 0$

$t_{xmin} \leq t \leq t_{xmax}$

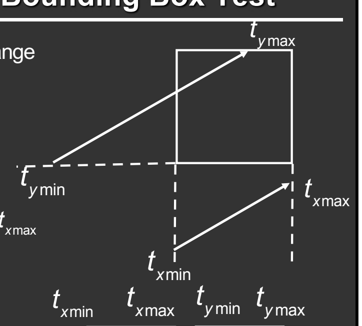
$t_{ymin} \leq t \leq t_{ymax}$

if $t_{xmin} > t_{ymax}$ *OR* $t_{ymin} > t_{xmax}$

return false;

else

return true;

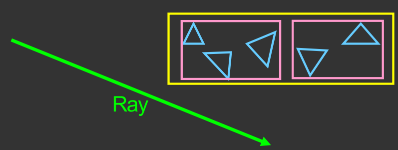


No intersection if x and y ranges don't overlap

84

Bounding Box Test

- Ray-Intersection is simple coordinate check
- Intricacies with test, see Shirley book
- Hierarchical Bounding Boxes



85

Hierarchical Bounding Box Test

- If ray hits root box
  - Intersect left subtree
  - Intersect right subtree
  - Merge intersections (find closest one)
- Standard hierarchical traversal
  - But caveat, since bounding boxes may overlap
- At leaf nodes, must intersect objects

86

Creating Bounding Volume Hierarchy

```
function bvh-node::create (object array A, int AXIS)
    N = A.length() ;
    if (N == 1) {left = A[0]; right = NULL; bbox = bound(A[0]);}
    else if (N == 2) {
        left = A[0] ; right = A[1] ;
        bbox = combine(bound(A[0]),bound(A[1])) ;
    else
        Find midpoint m of bounding box of A along AXIS
        Partition A into lists of size k and N-k around m
        left = new bvh-node (A[0…k],(AXIS+1) mod 3) ;
        right = new bvh-node(A[k+1…N-1],(AXIS+1) mod 3);
        bbox = combine (left -> bbox, right -> bbox) ;
```

From page 305 of Shirley book

87

Area Heuristics

- Instead of mid-point of bounding box, alternating axes, pick the axis and the location to split carefully
- The algorithm can test several splitting planes (at least 9 recommended) across x,y,z and chooses best one
- Area Heuristic: min $a_1 n_1 + a_2 n_2$ considering areas of each child box and number of primitives contained in each
- Longer for construction but better balanced
- Ideally speeds up raytracing *(in Optix BVH built in)*
- (Optional, but if interested read up on Surface Area Heuristic [SAH] and similar methods. Also see fast updates for animations, dynamic scenes)

88

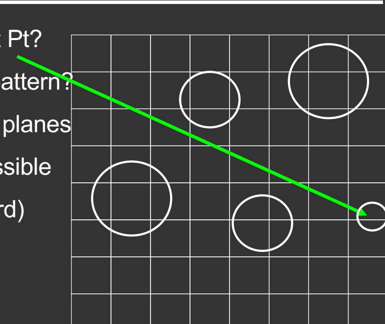
Uniform Spatial Subdivision

- Different idea: Divide space rather than objects
- In BVH, each object is in one of two sibling nodes
  - A point in space may be inside both nodes
- In spatial subdivision, each space point in one node
  - But object may lie in multiple spatial nodes
- Simplest is uniform grid (have seen this already)
- Challenge is keeping all objects within cell
- And in traversing the grid

89

Traversal of Grid High Level

- Next Intersect Pt?
- Irreg. samp. pattern?
- But regular in planes
- Fast algo. possible
- (more on board)



90

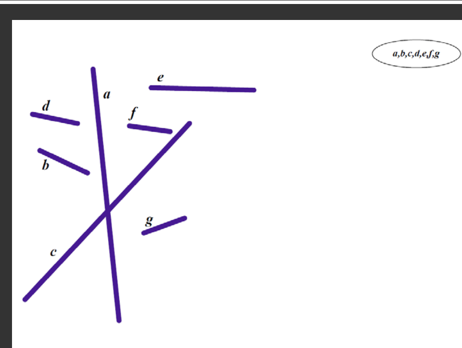
BSP Trees

- Used for visibility and ray tracing
  - Book considers only axis-aligned splits for ray tracing
  - Sometimes called kd-tree for axis aligned
- Split space (binary space partition) along planes
- Fast queries and back-to-front (painter's) traversal
- Construction is conceptually simple
  - Select a plane as root of the sub-tree
  - Split into two children along this root
  - Random polygon for splitting plane (may need to split polygons that intersect it)

BSP slides courtesy Prof. O'Brien

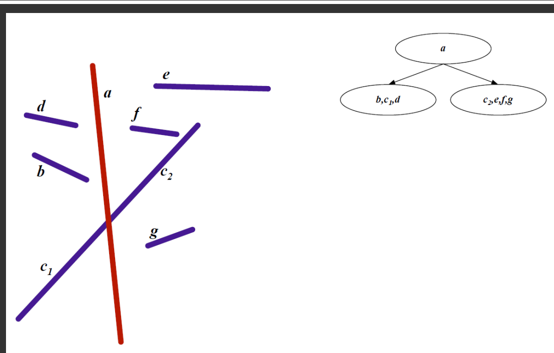
91

Initial State



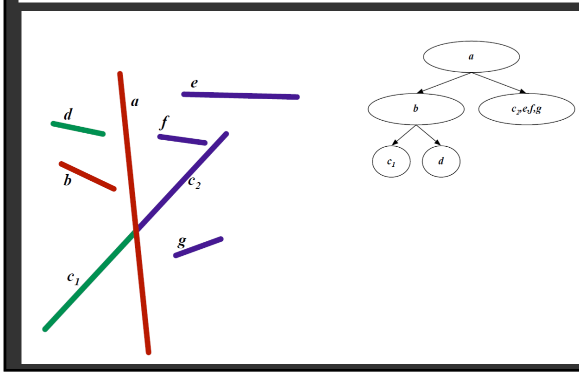
92

First Split



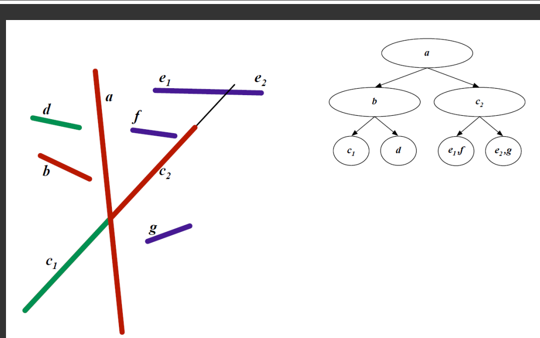
93

Second Split



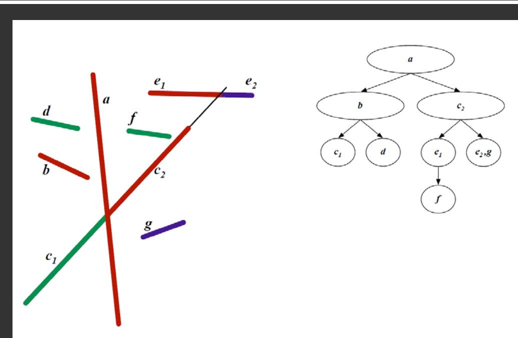
94

Third Split



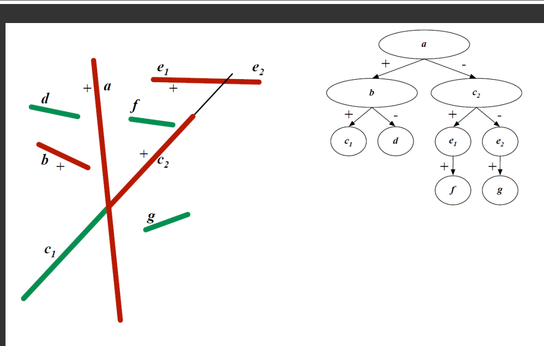
95

Fourth Split



96

Final BSP Tree



97

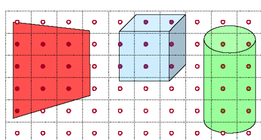
BSP Trees Cont’d

- Continue splitting until leaf nodes
- Visibility traversal in order
  - Child one
  - Root
  - Child two
- Child one chosen based on viewpoint
  - Same side of sub-tree as viewpoint
- BSP tree built once, used for all viewpoints

98

Other Accelerations

- Screen space coherence
  - Check last hit first
  - Beam tracing
  - Pencil tracing
  - Cone tracing
- Memory coherence
  - Large scenes
- Parallelism
  - Ray casting is “embarassingly parallelizable”
- etc.

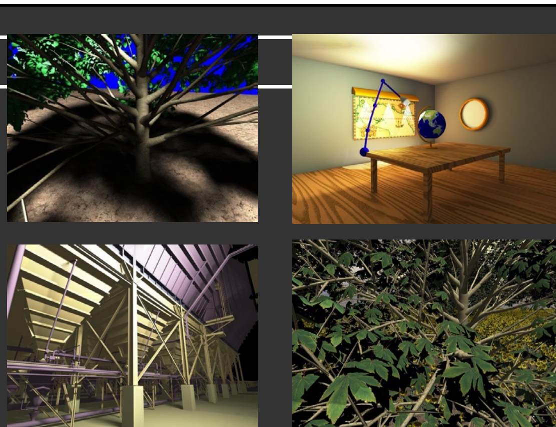


99

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Shadows / Reflections (core algorithm)
- Optimizations
- *Current Research*

100



101

Interactive Raytracing

- Ray tracing historically slow
- Now viable alternative for complex scenes
  - Key is sublinear complexity with acceleration; need not process all triangles in scene
- Allows many effects hard in hardware
- Today graphics hardware and software (NVIDIA Optix 6, RTX chips 10G+ rays per second). Video
- Tiger Demo (NVIDIA; see slide)

102

Today: Real-Time Denoising at 1spp

NVIDIA Tiger Demo 2021

103

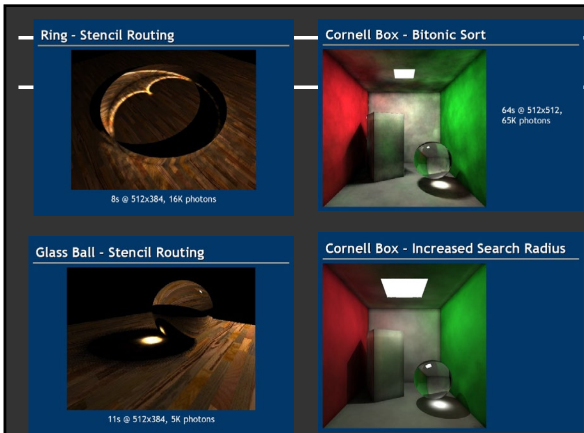
Raytracing on Graphics Hardware

- Modern Programmable Hardware general streaming architecture
- Can map various elements of ray tracing
- Kernels like eye rays, intersect etc.
- In vertex or fragment programs
- Convergence between hardware, ray tracing

[Purcell et al. 2002, 2003]

http://graphics.stanford.edu/papers/photongfx

104



105