Ray Tracing

Reading

Foley *et al.*, 16.12

Optional:
- Glassner, An introduction to Ray Tracing, Academic Press, Chapter 1.
- T. Whitted. “An improved illumination model for shaded display”. *Communications of the ACM*} 23(6), 343-349, 1980.

What is light

Descartes (ca. 1630)
- Light is a pressure phenomenon in the ``plenum''

Hooke (1665)
- Light is a rapid vibration -- first wave theory

Newton (1666)
- Refraction experiment revealed rectilinear propagation
- Light is a particle (corpuscular theory)

Young (1801)
- Two slit experiment
- Light is a wave

Maxwell (ca. 1860)
- Light is an electromagnetic disturbance

Einstein (1905)
- Light comes in quanta -- photons

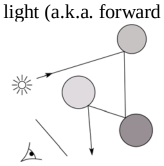
Modern theory: wave-particle duality.

Geometric optics

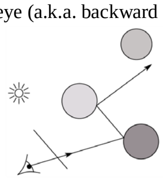
We will take the view of **geometric optics**
- Light is a flow of photons with wavelengths. We'll call these flows ``light rays.''
- Light rays travel in straight lines in free space.
- Light rays do not interfere with each other as they cross.
- Light rays obey the laws of reflection and refraction.
- Light rays travel form the light sources to the eye, but the physics is invariant under path reversal (reciprocity).

Eye vs. Light

- Starting at the light (a.k.a. forward ray tracing, photon tracing)



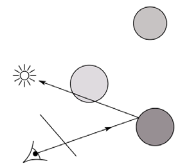
- Starting at the eye (a.k.a. backward ray tracing)



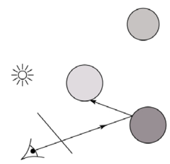
5

Hybrid methods

Local illulmination
- Cast one ray, shade according to light



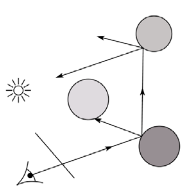
Appel (1968)
- Cast one eye ray & one ray to light



6

Whitted (1980)

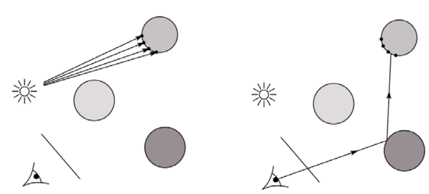
Eye ray tracing and rays to light & recursive ray tracing



7

Heckbert (1990)

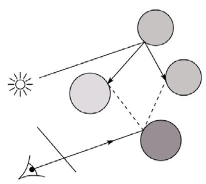
Ray tracing & light ray tracing & light storage on surface



8

Veach (1995)

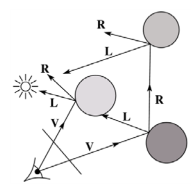
- Eye ray tracing & light ray tracing & path connection



9

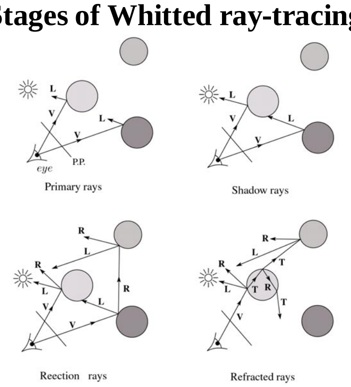
Whitted ray-tracing algorithm

1. For each pixel, trace a **primary ray** to the first visible surface
2. For each intersection trace **secondary rays**:
   - **Shadow rays** in directions Li to light sources
   - **Reflected ray** in direction R
   - **Refracted ray** (**transmitted ray**) in direction T



10

Stages of Whitted ray-tracing



11

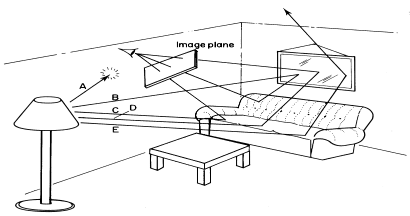
Why Ray Tracing?

- So far, we can do **ray casting**: for each pixel in the projection plane, find the object visible at that pixel and apply your favorite shading model.
- What does this model miss?

12

Forward Ray Tracing

- Rays emanate from light sources and bounce around in the scene.
- Rays that pass through the projection plane and enter the eye contribute to the final image.



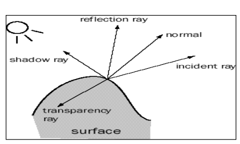
- What's wrong with this method?

Backward Ray Tracing

- Rather than propagating rays indiscriminately from light sources, we'd like to ask "which rays will definitely contribute to the final image?"
- We can get a good approximation of the answer by firing rays from the eye, through the projection plane and into the scene
  - These are the paths that light must have followed to affect the image

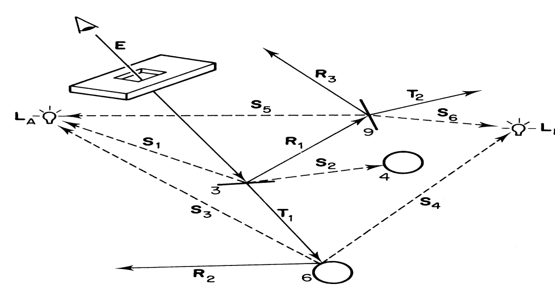
Kinds of Rays

- A ray that leaves the eye and travels out to the scene is called a **primary ray**.
- When a ray hits an object, we spawn three new (backward) rays to collect light that must contribute to the incoming primary ray:
  - **Shadow rays** to light sources, used to attenuate incoming light when applying the shading model
  - **Reflection rays**, which model light bouncing off of other surfaces before hitting this surface
  - **Transparency rays**, which model light refracting through the surface before leaving along the primary ray

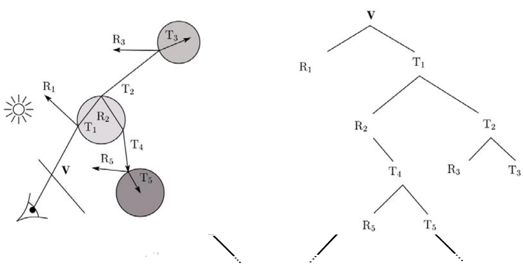


- Shadow rays stop at light sources, but reflection and transparency rays behave just like primary rays!

Example of Ray Tracing



The Ray Tree



Shading

If I(P, **u**) is the intensity seen from point P along direction **u**

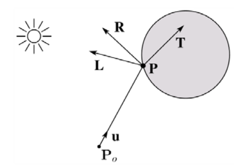
$$I(P,\mathbf{u}) = I_{direct} + I_{reflected} + I_{transmitted}$$

where

I_{direct} is computed from the Phong model (next lecture)

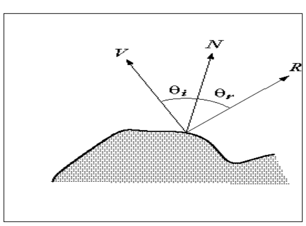
$I_{reflected} = k_s I(P,\mathbf{R})$

$I_{transmitted} = k_t I(P,\mathbf{T})$

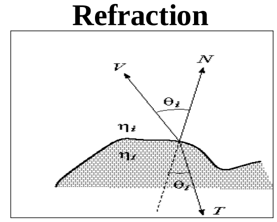


Reflection

- Reflected light from objects behaves like specular reflection from light sources
  - Reflectivity is just specular color
  - Reflected light comes from direction of perfect specular reflection



Refraction

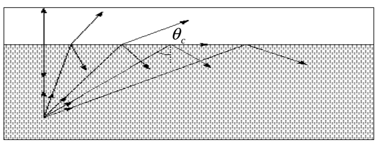


- Amount to transmit determined by transparency coefficient, which we store explicitly
- *T* comes from Snell's law

$$\eta_i \sin(\theta_i) = \eta_t \sin(\theta_t)$$

Total Internal Reflection

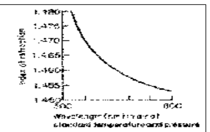
- When passing from a dense medium to a less dense medium, light is bent further away from the surface normal
- Eventually, it can bend right past the surface!
- The θ_i that causes θ_t to exceed 90 degrees is called the **critical angle (θ_c)**. For θ_i greater than the critical angle, no light is transmitted.
- A check for TIR falls out of the construction of T



Index of Refraction

- Real-world index of refraction is a complicated physical property of the material

Medium	Index of refraction
Vaccum	1
Air	1.0003
Water	1.33
Fused quartz	1.46
Glass, crown	1.52
Glass, dense flint	1.66
Diamond	2.42



Index of refraction variation for fused quartz

- IOR also varies with wavelength, and even temperature!
- How can we account for wavelength dependence when ray tracing?

Parts of a Ray Tracer

- What major components make up the core of a ray tracer?

Ray Tracing Pseudocode

```
color trace( point P0, direction D )
{
        (P,Oi) = intersect( P0, D );
        I = 0
        for each light source l {
                (P', LightObj) = intersect(P, dir(P,l))
                if LightObj = l {
                        I = I + I(l)
                }
        }
        I = I + Obj.Ks * trace(P, R)
        I = I + Obj.Kt * trace(P, T)
        return I
}
```

Controlling Tree Depth

- Ideally, we'd spawn child rays at every object intersection forever, getting a "perfect" color for the primary ray.
- In practice, we need heuristics for bounding the depth of the tree (i.e., recursion depth)
- ?

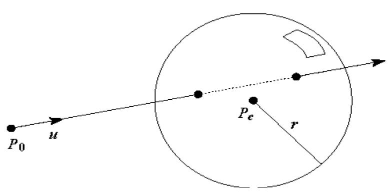
25

Ray-Object Intersection

- Must define different intersection routine for each primitive
- The bottleneck of the ray tracer, so make it fast!
- Most general formulation: find all roots of a function of one variable
- In practice, many optimized intersection tests exist (see Glassner)

26

Ray-Sphere Intersection

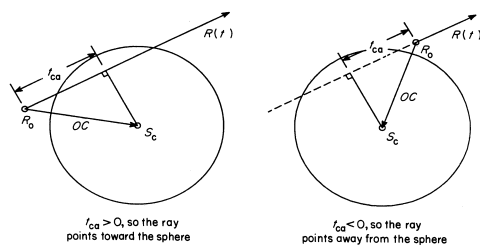


- Given a sphere centered at P_c =[0,0,0] with radius r and a ray $P(t) = P_0 + t\boldsymbol{u}$, find the intersection(s) of $P(t)$ with the sphere.

27

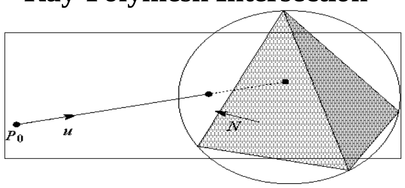
Fast Failure

- We can greatly speed up ray-object intersection by identifying cheap tests that guarantee failure
- Example: if origin of ray is outside sphere and ray points away from sphere, fail immediately.



28

Ray-Polymesh Intersection



1. Use bounding sphere for fast failure
2. Test only front-facing polygons
3. Intersect ray with each polygon's supporting plane
4. use a point-in-polygon test
5. Intersection point is smallest *t*

Object hierarchies and ray intersection

How do we intersect with primitives transformed with affine transformations?

Numerical Error

- Floating-point roundoff can add up in a ray tracer, and create unwanted artifacts
  - Example: intersection point calculated to be ever-so-slightly *inside* the intersecting object. How does this affect child rays?
- Solutions:
  - Perturb child rays
  - Use global ray epsilon

Goodies

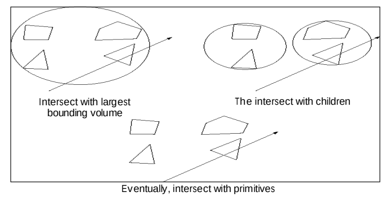
- There are some advanced ray tracing feature that self-respecting ray tracers shouldn't be caught without:
  - Acceleration techniques
  - Antialiasing
  - CSG
  - Distribution ray tracing

Acceleration Techniques

- Problem: ray-object intersection is very expensive
  - make intersection tests faster
  - do fewer tests

33

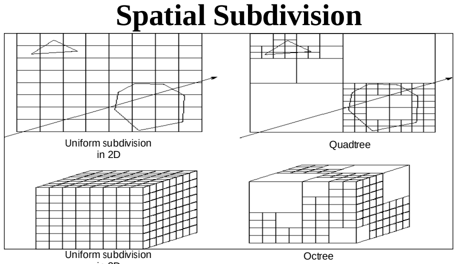
Hierarchical Bounding Volumes



- Arrange scene into a tree
  - Interior nodes contain primitives with very simple intersection tests (e.g., spheres). Each node's volume contains all objects in subtree
  - Leaf nodes contain original geometry
- Like BSP trees, the potential benefits are big but the hierarchy is hard to build

34

Spatial Subdivision

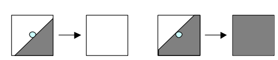


- Divide up space and record what objects are in each cell
- Trace ray through **voxel** array

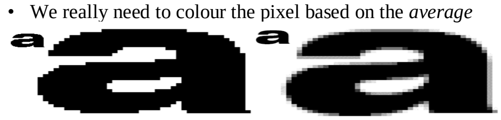
35

Antialiasing

- So far, we have traced one ray through each pixel in the final image. Is this an adequate description of the contents of the pixel?



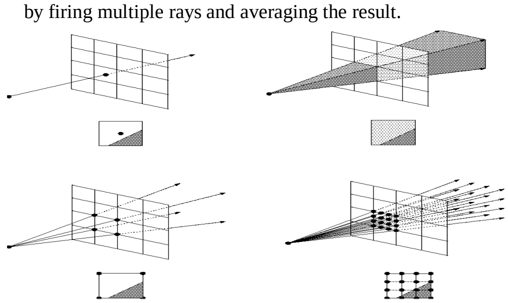
- This quantization through inadequate sampling is a form of **aliasing**. Aliasing is visible as "jaggies" in the ray-traced image.
- We really need to colour the pixel based on the *average*



36

Supersampling

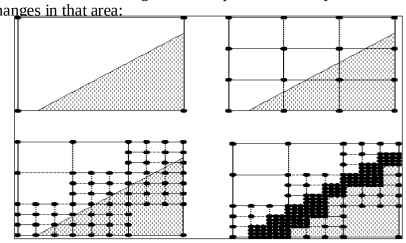
- We can approximate the average colour of a pixel's area by firing multiple rays and averaging the result.



37

Adaptive Sampling

- Uniform supersampling can be wasteful if large parts of the pixel don't change much.
- So we can subdivide regions of the pixel's area only when the image changes in that area:

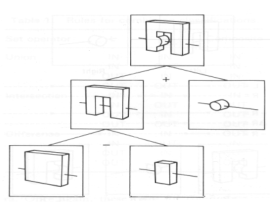


- How do we decide when to subdivide?

38

CSG

- CSG (constructive solid geometry) is an incredibly powerful way to create complex scenes from simple primitives.

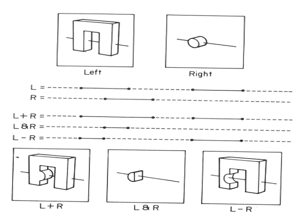


- CSG is a modeling technique; basically, we only need to modify ray-object intersection.

39

CSG Implementation

- CSG intersections can be analyzed using "Roth diagrams".
  - Maintain description of *all intersections* of ray with primitive
  - Functions to combine Roth diagrams under CSG operations

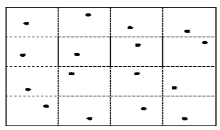


- An elegant and extremely slow system

40

Distribution Ray Tracing

- Usually known as "distributed ray tracing", but it has nothing to do with distributed computing
- General idea: instead of firing one ray, fire multiple rays in a jittered grid



- Distributing over different dimensions gives different effects
- Example: what if we distribute rays over pixel area?

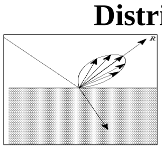
41

Disrtibuted ray tracing pseudocode

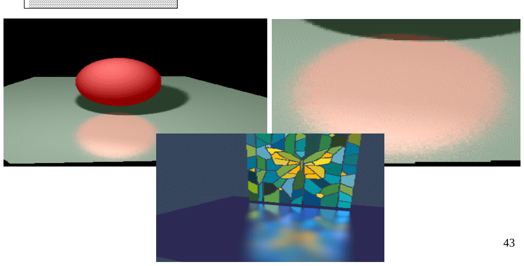
1. Partition pixel into 16 regions assigning them id 1-16
2. Partition the reflection direction into 16 angular regions and assign an id (1-16) to each
3. Select sub pixel m=1
4. Cast a ray through m, jittered within its region
5. After finding an intersection, reflect into sub-direction m, jittered within that region
6. Add result to current pixel total
7. Increment m and if m<= 16, go to step 4
8. Divide by 16, store result and move on to next pixel.

42

Distributing Reflections



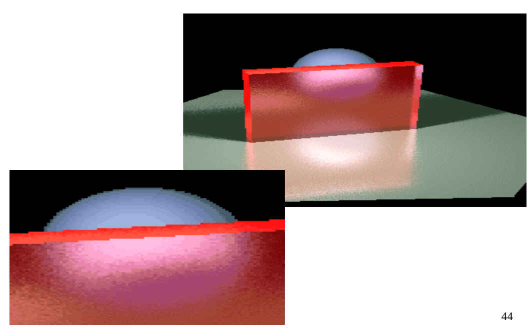
- Distributing rays over reflection direction gives:



43

Distributing Refractions

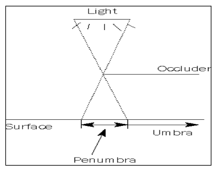
- Distributing rays over transmission direction gives:

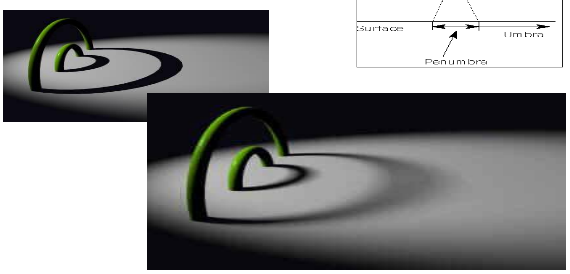


44

Distributing Over Light Area

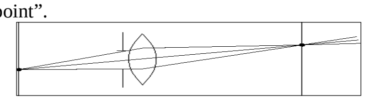
- Distributing over light area gives:

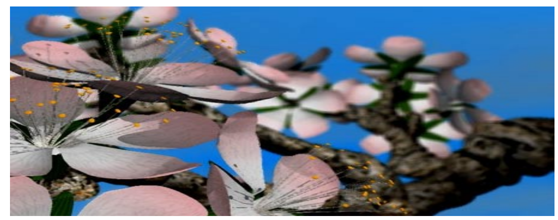




Distributing Over Aperature

- We can fake distribution through a lens by choosing a point on a finite aperature and tracing through the “in-focus point”.





Distributing Over Time

- We can endow models with velocity vectors and distribute rays over *time*. this gives:

47