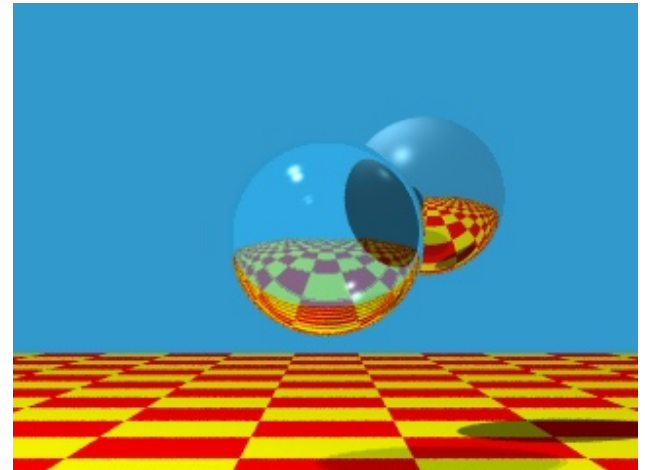


Ray tracing

Computer Graphics
COMP 770 (236)
Spring 2007

Instructor: Brandon Lloyd



From last time...

■ Hidden surface removal

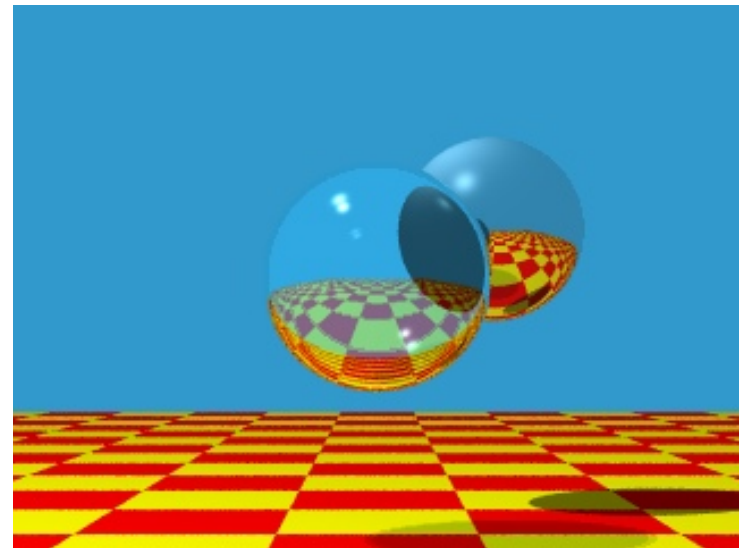
- Painter's algorithm
- Clipping algorithms
- Area subdivision
- BSP trees
- Z-Buffer
- Ray casting

Today's topics

- Recursive ray tracing
- Intersection tests
 - Plane
 - Polygon
 - Sphere

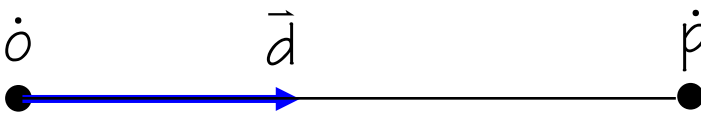
Recursive ray casting

- Ray casting generally dismissed early on:
 - Takes no advantage of screen space coherence
 - Requires costly visibility computation
 - Only works for solids
 - Forces per pixel illumination evaluations
 - Not suited for immediate mode rendering
- Gained popularity in when Turner Whitted (1980) recognized that *recursive* ray casting could be used for global illumination effects



Generalized ray casting

- Ray casting used to compute visibility at the eye
- Visibility for arbitrary points needed for shading
 - Reflections
 - Refraction and transparency
 - Shadows
- We need to compute the first surface hit along a ray
 - represent ray with origin and direction
 - compute intersections of objects with ray
 - return closest object

$$\dot{p}(t) = \dot{o} + t\vec{d}$$


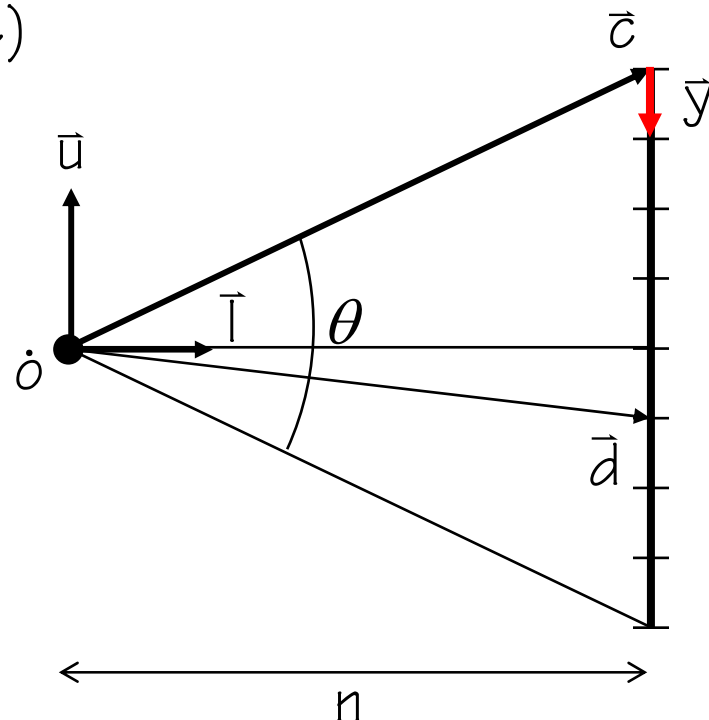
The diagram illustrates a ray in 2D space. It consists of a horizontal line with two black dots representing points. The left dot is labeled \dot{o} and the right dot is labeled \dot{p} . A blue arrow points from \dot{o} to \dot{p} , and is labeled \vec{d} above it.

Generating primary rays

$$\bar{x} = \frac{n \tan(\theta/2) \text{width/height}}{\text{width}} \bar{r} \quad \bar{y} = \frac{n \tan(\theta/2)}{\text{height}} \bar{u}$$

$$\bar{c} = \left(-\frac{\text{width}}{2} + 0.5\right) \bar{x} + \left(-\frac{\text{height}}{2} + 0.5\right) \bar{y} + n \bar{1}$$

$$\bar{d} = \bar{c} + i\bar{x} + j\bar{y} \quad i \in [0, \text{width}), j \in [0, \text{height})$$

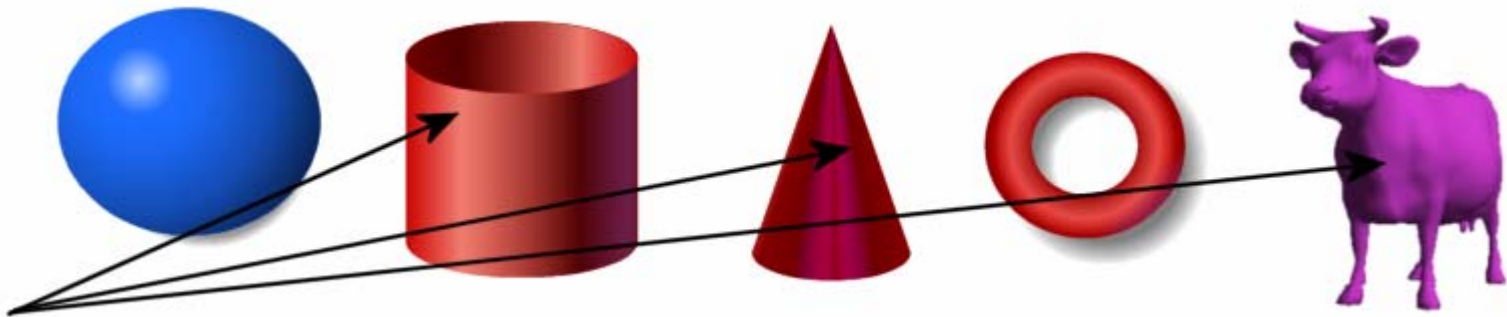


Generating secondary rays

- The origin is the intersection point p_0
- Direction depends on the type of ray
 - shadow rays – use direction to the light source
 - reflection rays – use incoming direction and normal to compute reflection direction
 - transparency – continue with same direction
 - refraction – use snell's law

Intersection tests

Go through all of the objects in the scene to determine the one closest to the origin of the ray (the eye).



Strategy: Solve for the intersection of the Ray with a mathematical description of the object.

Simple strategy

- Parametric ray equation

- Gives all points along the ray as a function of the parameter

$$\dot{p}(t) = \dot{o} + t\vec{d}$$

- Implicit surface equation

- Describes all points on the surface as the zero set of a function

$$f(\dot{p}) = 0$$

- Substitute ray equation into surface function and solve for t

$$f(\dot{o} + t\vec{d}) = 0$$

Ray plane intersection

- Implicit equation of a plane:

$$\vec{n} \cdot \vec{p} - d = 0$$

- Substitute ray equation:

$$\vec{n} \cdot (\vec{o} + t\vec{d}) - d = 0$$

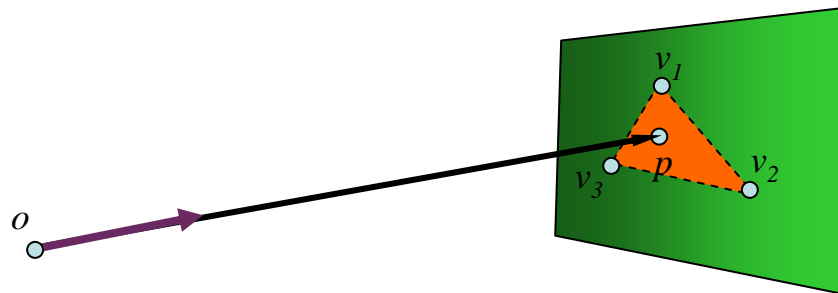
- Solve for t:

$$t(\vec{n} \cdot \vec{d}) = d - \vec{n} \cdot \vec{o}$$

$$t = \frac{d - \vec{n} \cdot \vec{o}}{\vec{n} \cdot \vec{d}}$$

Generalizing to Triangles

- Find of the point of intersection on the plane containing the triangle
- Determine if the point is inside the triangle
 - barycentric coordinate method
 - half-plane method



Barycentric coordinates

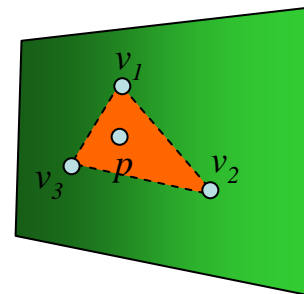
- Points in a triangle have positive barycentric coordinates:

$$\dot{p} = \alpha_1 \dot{v}_0 + \alpha_2 \dot{v}_1 + \alpha_3 \dot{v}_2 \quad \alpha_1 + \alpha_2 + \alpha_3 = 1$$

Expressed as a matrix equation:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} v_{1x} & v_{2x} & v_{3x} \\ v_{1y} & v_{2y} & v_{3y} \\ v_{1z} & v_{2z} & v_{3z} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}$$

$$\dot{p} = \mathbf{V} \vec{\alpha}$$
$$\vec{\alpha} = \mathbf{V}^{-1} \dot{p}$$



\mathbf{V}^{-1} can be precomputed to accelerate this calculation



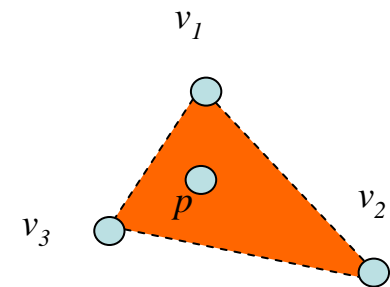
Barycentric coordinates

- Since we only care about the signs of the barycentric coordinates, we don't have to compute a complete inverse

$$\mathbf{V}^{-1} = \frac{\text{Adjoint}(\mathbf{V})}{\text{Det}(\mathbf{V})} \quad \text{Adjoint}(\mathbf{V}) = \begin{bmatrix} (\dot{v}_2 \times \dot{v}_3)^T \\ (\dot{v}_3 \times \dot{v}_1)^T \\ (\dot{v}_1 \times \dot{v}_2)^T \end{bmatrix}$$

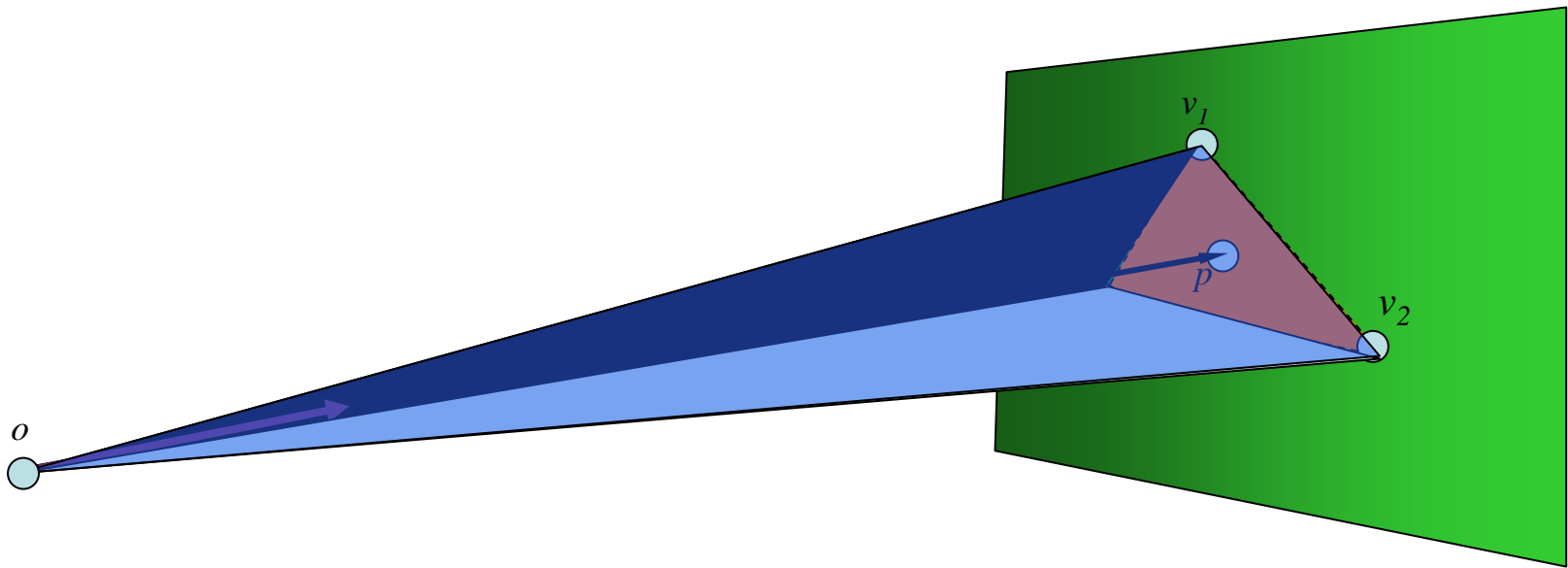
Issues

- Numerical robustness. There are lots of places that errors could creep in.
 - What if the point p is slightly off the plane?
 - What if the weights don't sum exactly to 1? Could be caused by p being off the plane as above, or by slight errors in computing $\text{Adjoint}(\mathbf{V})$.
 - Manifest itself when triangle is viewed near a grazing angle.
- No advantage in computing the $\text{Adjoint}(\mathbf{V})$ rather than the full inverse if inverse is precomputed
 - Ray-tracing is, in general, dominated by ray-intersection test
 - It pays to precompute as much as possible
- Barycentric coordinates can be used for interpolating vertex parameters
 - normals, colors, texture coordinates, etc.



Half-Plane Method

- Test intersection point against planes connecting each triangle edge with the eye's position
- Closely related to the Adjoint(V) of barycentric coordinates but more general



Implementation Steps

- Compute plane equations:

$$\bar{n} \cdot \dot{p} - d = 0$$

$$\bar{n} = (\dot{v}_a - \dot{o}) \times (\dot{v}_b - \dot{o})$$

- Find d value:

$$\bar{n} \cdot \dot{o} = d$$

- Pick a plane orientation:

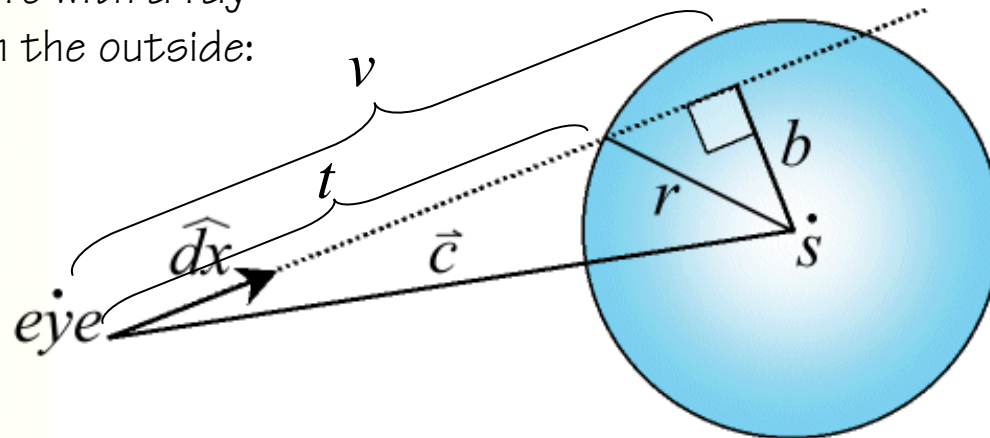
- Several choices here. The easiest is to evaluate p in the for the first plane, and make sure that all subsequent planes have the same sign. Assuming that you traverse the edges in a consistent order.

Tradeoffs of Half-Plane Method

- Can still preprocess all triangles, but the cross products depend on the viewing position
- Numerically more robust
 - including the eye as in our computation avoids problems when the intersection point lies slightly off the plane.
- Relation to the barycentric coordinate method
 - cross products of half plane method are the same as in Adjoint(\mathbf{V}) if the eye position is the origin

Spheres

Intersecting a sphere with a ray originating from the outside:



A sphere is defined by its center, s , and its radius r . The intersection of a ray with a sphere can be computed as follows:

A vector from the eye to the sphere's center.

$$\vec{c} = s - \text{eye}$$

$$v = \vec{c} \cdot \vec{dx}$$

$$b^2 = \vec{c} \cdot \vec{c} - v^2$$

Apply the Pythagorean law to find b

$$t = v \pm \sqrt{r^2 - b^2}$$

And once more to find t

How many "dx"-sized steps to the rays point of closest approach from the sphere's center.

Speeding up Intersection Test

■ Early Rejection

- Ray tracing dominated by intersection tests
- Try to reduce unnecessary work

■ Test for sphere within range of ray

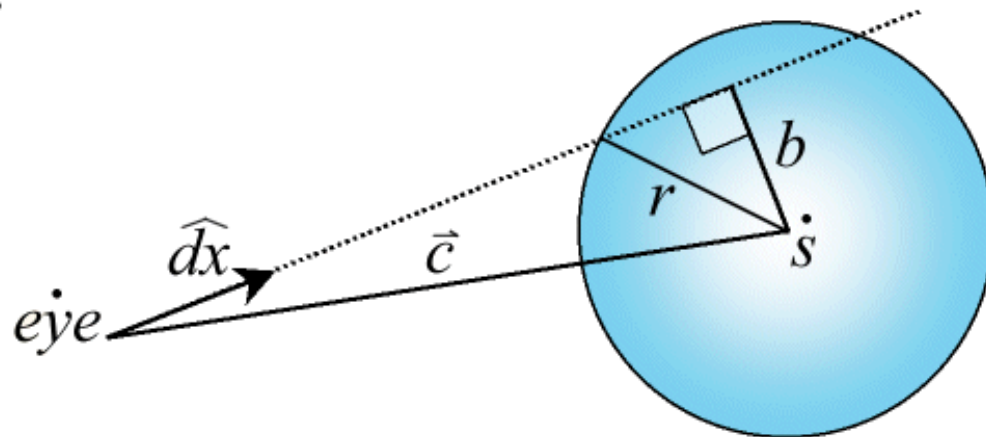
- When testing all the objects, we want the closest. At any point we usually have a t_{best} that is the closest intersection we have found so far.

$$\vec{c} = \dot{s} - eye$$

$$v = \hat{dx} \cdot \vec{c}$$

$$t > v - r$$

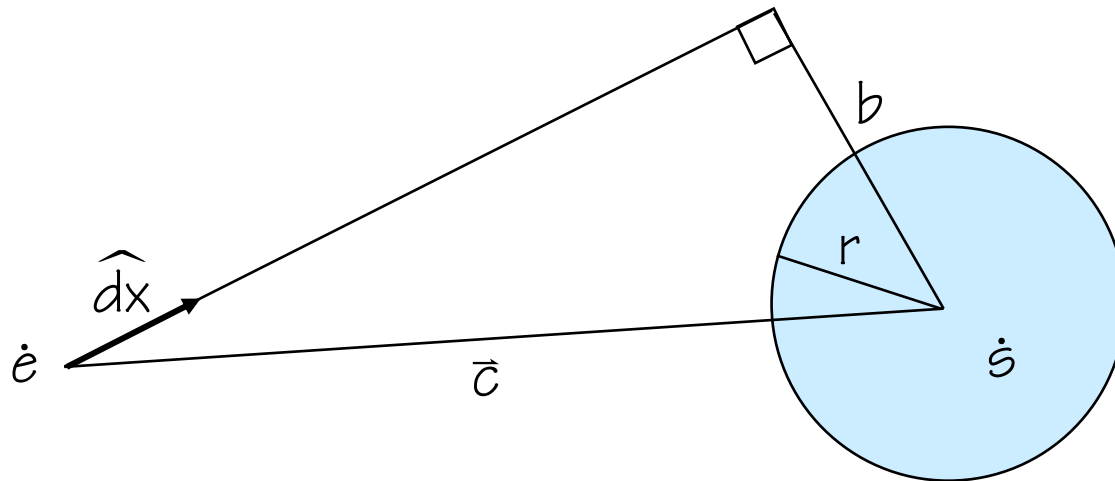
(why?)



More Trivial Rejections

- Test the radicand $r^2 - b^2 < 0$
 - Have to test anyway to avoid generating an exception
 - Means that the ray misses the sphere (no real roots)

$$t = v - \sqrt{r^2 - b^2}$$



Next time

- Robustness issues
- Optimizations
 - Acceleration structures
- Distribution ray tracing
 - anti-aliasing
 - depth of field
 - soft shadows
 - motion blur
- Volume effects
- Code structure