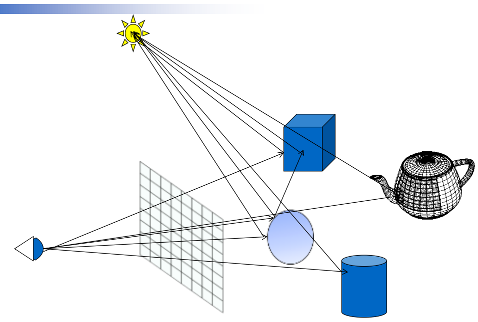
Raytracing: Intersections

COSC 4328/5327
Scott A. King

Backward Tracing

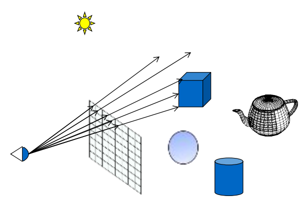


Basic Ray Casting Method

- $\forall$ pixels in screen
  - Shoot ray $\vec{p}$ from the eye through the pixel.
  - Find closest ray-object intersection.
  - Get color at intersection

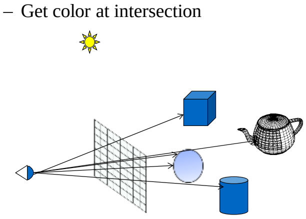
Basic Ray Casting Method

- $\forall$ pixels in screen
  - Shoot ray $\vec{p}$ from the eye through the pixel.
  - Find closest ray-object intersection.
  - Get color at intersection



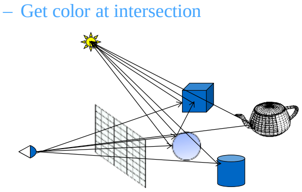
Basic Ray Casting Method

- $\forall$ pixels in screen
  - Shoot ray $\vec{p}$ from the eye through the pixel.
  - Find closest ray-object intersection.
  - Get color at intersection



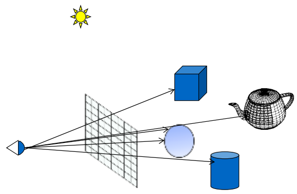
Basic Ray Casting Method

- $\forall$ pixels in screen
  - Shoot ray $\vec{p}$ from the eye through the pixel.
  - Find closest ray-object intersection.
  - Get color at intersection



Basic Ray Casting Method

- ∀ pixels in screen
  - Shoot ray $\vec{p}$ from the eye through the pixel.
  - Find closest ray-object intersection.
  - Get color at intersection

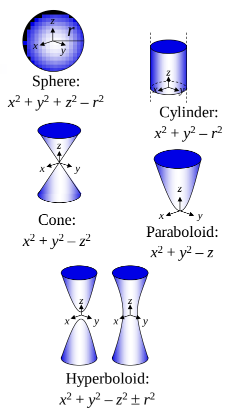


The Truth!

- Solving intersections can be hard
- Simple surfaces can yield a closed-form solution
- General case: non-linear root finding
  - No simple, quick method.
  - Expensive!
  - Won't always converge
  - When repeated for millions of rays, you WILL find the divergent case!

Good News

- Use primitives with closed-form solutions.
- Use object-oriented methods.
  - 1 intersection method per primitive type.
  - Object does its own intersecting.
- Surfaces with closed-form solutions.
  - quadrics: sphere, cylinder, cone, ellipsoid, paraboloid, etc.
  - polygons.
  - tori, super-quadrics, low-order splines.



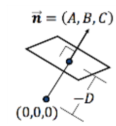
Ray-Object Intersection

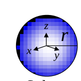
- Define object implicitly by a function $f(P) = 0$
  - For any point P, when f is 0, the point is on the surface,
  - non-zero defines how far away from the surface you are,
    - usually negative below surface (inside object)
- Many objects can be defined implicitly
  - Give potentially infinite resolution
  - Tessellating objects harder than using f directly
- An infinite plane is defined by the function:

  $$f(x,y,z) = Ax + By + Cz + D$$

- A sphere of radius R in 3-space:

  $$f(x,y,z) = x^2 + y^2 + z^2 - R^2$$





Basic Ray Model

- Let's treat a ray as a vector. Namely we can represent a ray by the vector form:

  $$\vec{p} = \vec{u} + \vec{v}t$$

Basic Ray Model

- Let's treat a ray as a vector. Namely we can represent a ray by the vector form:

  $$\vec{p} = \vec{u} + \vec{v}t$$

- where:

  $\vec{p}$ is any point along the ray

Basic Ray Model

- Let's treat a ray as a vector. Namely we can represent a ray by the vector form:

$$\vec{p} = \vec{u} + \vec{v}t$$

- where:

  $\vec{p}$ is any point along the ray

  $\vec{u}$ is the starting point

Basic Ray Model

- Let's treat a ray as a vector. Namely we can represent a ray by the vector form:

$$\vec{p} = \vec{u} + \vec{v}t$$

- where:

  $\vec{p}$ is any point along the ray

  $\vec{u}$ is the starting point

  $\vec{v}$ **(unit vector)** is the direction

Basic Ray Model

- Let's treat a ray as a vector. Namely we can represent a ray by the vector form:

$$\vec{p} = \vec{u} + \vec{v}t$$

- where:

  $\vec{p}$ is any point along the ray

  $\vec{u}$ is the starting point

  $\vec{v}$ (**unit vector**)is the direction

  t is distance along ray.

```
struct ray {
    vec4 start;
    vec4 direction;
}
```

Ray/Sphere Intersection

- Simple Case: A sphere of radius 1 centered at the origin

$$x^2 + y^2 + z^2 = 1$$
$$\vec{p}^2 = 1$$

- The ray $\vec{p}$ intersects the sphere when $\vec{p} = \vec{u} + \vec{v}t$ satisfies the equation for the sphere.

$$\vec{u}^2 + 2\vec{u}\vec{v}t + \vec{v}^2t^2 = 1$$

- We solve using the quadratic formula.
  - See the ray tracing notes for details on avoiding round-off errors and solving efficiently.

$$\underbrace{\boxed{\vec{u}^2 - 1}}_{c} + \underbrace{\boxed{2\vec{u}\vec{v}}}_{b}t + \underbrace{\boxed{\vec{v}^2}}_{a}t^2 = 0$$

Ray/Sphere Intersection

- What about a sphere centered at (c_x, c_y, c_z) or radius r?

$$(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 = r^2 \quad \boxed{= (\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c})}$$

- Plug in ray equation and get

$$(u_x + v_xt - c_x)^2 + (u_y + v_yt - c_y)^2 + (u_z + v_z t - c_z)^2 = r^2$$

- And solve using the quadratic formula.

$$\boxed{= (\vec{u} + \vec{v}t - \vec{c})^2}$$

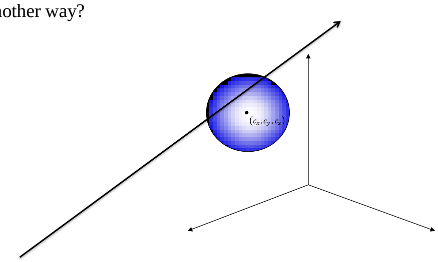
$$a = v_x{}^2 + v_y{}^2 + v_z{}^2 = 1$$

$$b = 2\left(v_x(u_x - c_x) + v_y(u_y - c_y) + v_z(u_z - c_z)\right) \quad \boxed{= 2(\vec{u} - \vec{c}) \cdot \vec{v}}$$

$$c = (u_x - c_x)^2 + (u_y - c_y)^2 + (u_z - c_z)^2 - r^2 \quad \boxed{= (\vec{u} - \vec{c}) \cdot (\vec{u} - \vec{c}) \text{-} r^2}$$

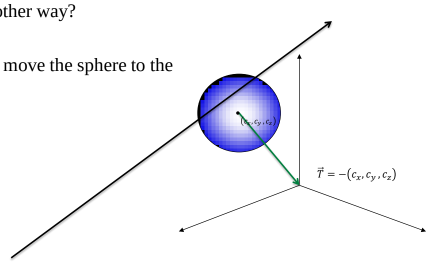
Ray/Sphere Intersection

- Is there another way?



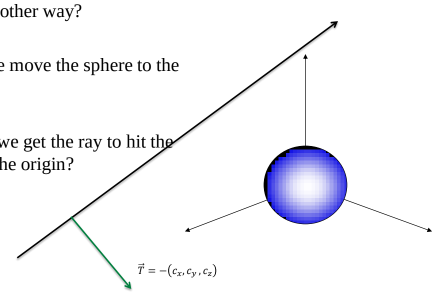
Ray/Sphere Intersection

- Is there another way?
- What if we move the sphere to the origin?



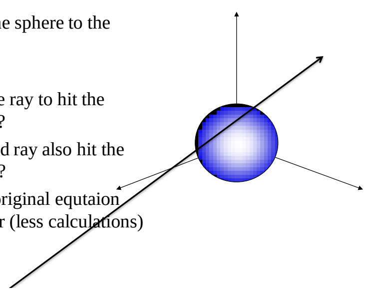
Ray/Sphere Intersection

- Is there another way?
- What if we move the sphere to the origin?
- How will we get the ray to hit the sphere at the origin?



Ray/Sphere Intersection

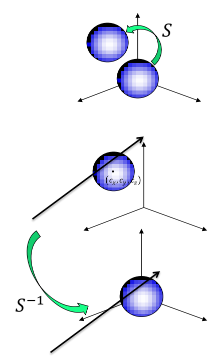
- Is there another way?
- What if we move the sphere to the origin?
- How will we get the ray to hit the sphere at the origin?
- Will the transformed ray also hit the transformed sphere?
- So we can use the original equtaion which is a it simpler (less calculations)



Ray/Sphere Intersection

- This approach works for any transformed object.
- Start with a primitive object. It is transformed (scaled, rotated translated, etc.) by a matrix, S. The inverse of that matrix will put it back to its original state.
- So the ray just needs to be transformed by S^{-1} then the simple ray/object intersection can be used.

$$\vec{u}' = S^{-1}\vec{u} \qquad \vec{v}' = S^{-1}\vec{v}$$



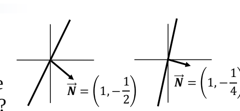
Intersection in World or Object Space?

- Sphere at origin (object space)
  $a = \vec{v}^2 = 1$
  $b = 2\vec{u}\vec{v}$
  $c = \vec{u}^2 - 1$
- Sphere centered at (c_x, c_y, c_z) with radius r
  $a = \vec{v}^2 = 1$
  $b = 2(\vec{u} - \vec{c})\cdot\vec{v}$
  $c = (\vec{u} - \vec{c})\cdot(\vec{u} - \vec{c}) - r^2$
- How much more math?
- What about extra math transforming ray?
- So why do it?
- How does ***t*** relate for object space ray and world space ray?

3-
1*

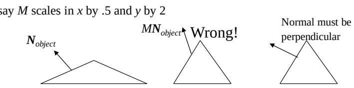
Normal

- We need the normal to calculate illumination and reflection vector.
- What is N for a unit sphere about the origin that intersects a ray at point p?
- What is N after that sphere is transformed using the matrix S?
- Can we transform the normal by S?
  - Rigid transforms fine (R,T)
  - Scales cause problems



Scaling distorts normal in opposite sense of scale applied to surface

Example: say M scales in x by .5 and y by 2



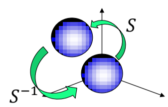
Review

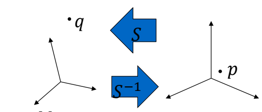
- The matrix S, transforms object space into world space
- Therefore the inverse goes from world space to object space
- If q is the corresponding point to p in world space, then

$$q = Sp$$
$$p_{world} = Sp_{object}$$
$$p_w = Sp_o$$

- Using the inverse

$$p = S^{-1}q$$
$$p_o = S^{-1}p_w$$
$$p_{object} = S^{-1}p_{world}$$





Normal

- For a plane that passes through the origin, and a point, p, on the plane,

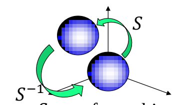
$$\vec{N} \cdot p = 0$$

- In matrix form this becomes, $\vec{N}^T p = 0$
- If q is the word space point to p

$$q = Sp \qquad p = S^{-1}q$$

- $\vec{N}^T S^{-1} q = 0$ describes a plane in world space whose normal is $\vec{N}^T S^{-1}$
- Let $\vec{N}_w$ (world space normal) be the normal of the transformed plane, so

$$\vec{N}_w = \vec{N}^T S^{-1}$$
$$\vec{N}_w = S^{-1T}\vec{N}$$



S goes from object space to world space

$a \bullet b = a^t\, b$

$\vec{N}^T S^{-1} q = 0$

So, the transpose of the inverse takes our object space normal into world space!

Ray/Triangle Intersection

- Triangle defined by vertices, a, b, c.
- 3 points defines a plane with a normal

$$\vec{n} = (b - a) \times (b - c)$$

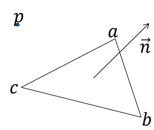
- For any point, p, in the plane

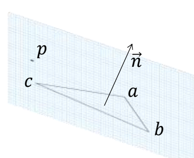
$$\vec{n} \cdot (p - b) = ?$$

- Where the ray intersects the plane, $\vec{p} = \vec{u} + \vec{v}t$ satisfies the above equation so

$$\vec{n} \cdot (\vec{u} + \vec{v}t - b) = 0$$
$$\vec{n} \cdot \vec{v}t = \vec{n} \cdot (\vec{u} - b)$$
$$t = \frac{\vec{n} \cdot (\vec{u} - b)}{\vec{n} \cdot \vec{v}}$$

top view



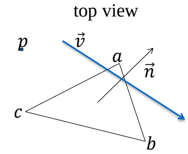


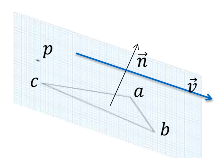
rotated view

Ray/Triangle Intersection

- If ray parallel to the plane, $\vec{n} \cdot \vec{v} = 0$
  - Can't solve for t and no intersection

top view



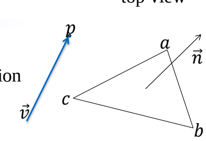


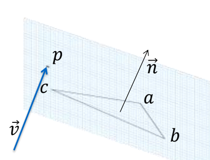
rotated view

Ray/Triangle Intersection

- If ray parallel to the plane, $\vec{n} \cdot \vec{v} = 0$
  - Can't solve for t and no intersection
- Otherwise we have an intersection within the plane.
  - Doesn't mean triangle is intersected.

top view





rotated view

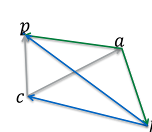
Ray/Triangle Intersection

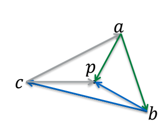
- If ray parallel to the plane, $\vec{n} \cdot \vec{v} = 0$
  - Can't solve for t and no intersection
- Otherwise we have an intersection within the plane.
  - Doesn't mean triangle is intersected.
- If the three dot products

$$(b - a) \times (p - a) \cdot \vec{n}$$
$$(c - b) \times (p - b) \cdot \vec{n}$$
$$(a - c) \times (p - c) \cdot \vec{n}$$

all have the same sign, the point is inside the triangle.

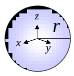
- Why?



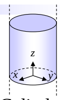


Quadrics

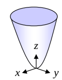
Ray: $x(t) = \mathbf{o}_x + t\,\mathbf{d}_x$,
$y(t) = \mathbf{o}_y + t\,\mathbf{d}_y$
$z(t) = \mathbf{o}_z + t\,\mathbf{d}_z$



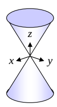
Sphere:
$x^2 + y^2 + z^2 - r^2$



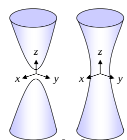
Cylinder:
$x^2 + y^2 - r^2$



Paraboloid:
$x^2 + y^2 - z$



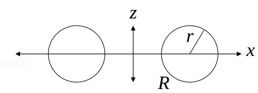
Cone:
$x^2 + y^2 - z^2$



Hyperboloid:
$x^2 + y^2 - z^2 \pm r^2$

Variations?
Use the transformation trick

Torus



- Product of two implicit circles

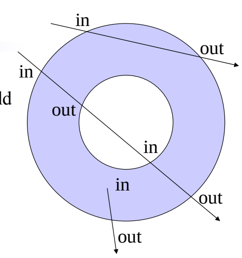
$$(x - R)^2 + z^2 - r^2 = 0$$
$$(x + R)^2 + z^2 - r^2 = 0$$
$$((x - R)^2 + z^2 - r^2)((x + R)^2 + z^2 - r^2)$$
$$= (x^2 - 2Rx + R^2 + z^2 - r^2)\,(x^2 + 2Rx + R^2 + z^2 - r^2)$$
$$= x^4 + 2x^2z^2 + z^4 - 2x2r^2 - 2z2r^2 + r^4 - 2x^2R^2 + 2z^2R^2 - 2r^2R^2 + R^4$$
$$= (x^2 + z^2 - r^2 - R^2)^2 + 4z^2R^2 - 4r^2R^2$$

- Surface of rotation: replace x^2 with $x^2 + y^2$
  $f(x,y,z) = (x^2 + y^2 + z^2 - r^2 - R^2)^2 + 4R^2(z^2 - r^2)$

- Quartic!!! (See Graphics Gems V for a solver)
- Up to four ray torus intersections

Ray-Object Intersection

- Returns intersection in a hit record
- "Next" field enables hit record to hold a list of intersections
- List only non-negative intersection parameters
- Ray always originates outside
  – If first $t = 0$ then ray originated inside
- Parity classifies ray segments
  – Odd segments "in"
  – Even segments "out"



Basic Ray Casting Method

- ∀ pixels in screen
  – Shoot ray $\vec{p}$ from the eye through the pixel.
  – Find closest ray-object intersection.
  – **Get color at intersection**

Illumination Model

Summary – putting it all together

Simple, non-recursive raytracer

```
P = eyePt

for each sample of image:
   Compute d
   for each object:
        Intersect ray P+td with object
//  Of all the objects that intersect ray, which one is
   visible?

   Select object with smallest non-negative t-value
   (visible object)

   For this object, find object space intersection point

   Compute normal at that point
   Transform normal to world space

   Use world space normal for lighting computations
```


Illumination Model

- For an object where does the light come from?
  – Direct from light source
  – Through the object.
  – Reflected from another object
  – Incident illumination (ambient)

Incident Illumination

- Where does this come from?
  - How about light transmitted (refracted) through another object.
  - How about light bouncing off of a non-reflective surface.
- For now we won't worry about this incident illumination, it is the subject of other methods (*global illumination*, *radiosity*, *photon mapping*). We'll just call it ambient light.

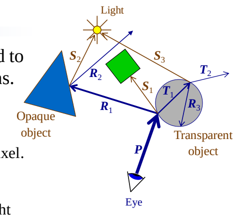
Types of Rays

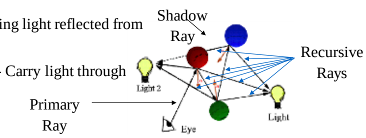
- To trace the light backward we need to perform the illumination calculations. To do this we need a few extra ray types.

  Primary rays - Carry light directly to a pixel.

  Secondary rays – get light to a point

  - **Shadow** rays - Bring light from the light source.
  - **Reflection** rays - Bring light reflected from another surface.
  - **Transmission** rays - Carry light through an object.





Recursive Ray Tracing

```
RayCast(screen)
for all pixels (x,y) in screen:
   trace(rayFromEyeThrough(x,y))
```

```
trace(ray)
  if (intersection= closestIntersection(ray))
    return Shade(intersection, ray)
  else return backgroundColor
```

```
closestIntersection(ray)
   for all objects find intersection
   for closest intersection return the intersection point, surface normal,
     surface, surface attributes, etc.
```

```
Shade(point, ray)
   Color = background;
   for each light
      if !Shadow(point, ray, light)
         Color += PhongIllumination(point, ray, light)
   if specularMaterial trace(reflect(point, ray))
   if refractive trace(refraction(point,ray))
   return Color
```

Demo (2d)

- http://www.siggraph.org/education/materials/HyperGraph/raytrace/rt_java/raytrace.html