

Lecture 11

More Ray Casting/Tracing

Basic Algorithm

For each pixel {

Shoot a ray from camera to pixel

for all objects in scene

 Compute intersection with ray

Find object with closest intersection

Display color using object + light property

}

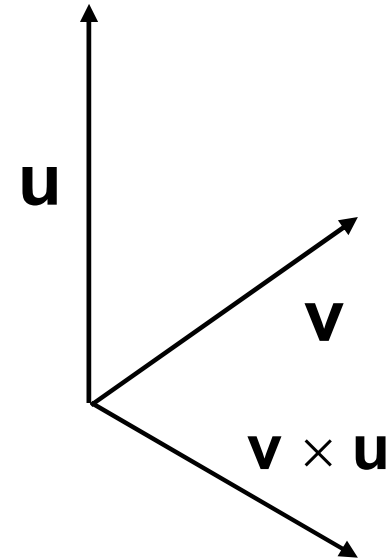
Pixel rays

Virtual world coordinates of pixel (i,j):
image center + displacements.

Image center: $\mathbf{c} + \mathbf{v}n$

pixel (i, j) = $\mathbf{c} + \mathbf{v}n +$

$$\left(\mathbf{h} - \left(\mathbf{j} + \frac{1}{2} \right) \frac{2\mathbf{h}}{\mathbf{M}} \right) \mathbf{u} + \left(\left(\mathbf{i} + \frac{1}{2} \right) \frac{2\mathbf{w}}{\mathbf{N}} - \mathbf{w} \right) \mathbf{v} \times \mathbf{u}$$



Basic Algorithm

For each pixel {

Shoot a ray from camera to pixel

for all objects in scene

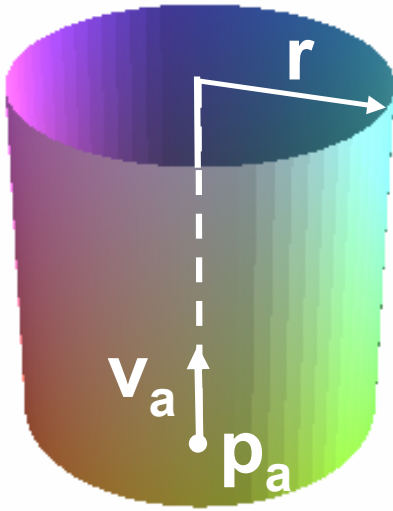
Compute intersection with ray

Find object with closest intersection

Display color using object + light property

}

Infinite cylinder-ray intersections



Infinite cylinder along y axis of radius r has equation $x^2 + z^2 - r^2 = 0$.

The equation for a more general cylinder of radius r oriented along a line $p_a + v_a t$:

$$(q - p_a - (v_a \cdot (q - p_a)) v_a)^2 - r^2 = 0$$

where $q = (x, y, z)$ is a point on the cylinder.

Infinite cylinder-ray intersections

To find intersection points with a ray $c+bt$
substitute $q = c+bt$ and solve:

$$(c - p_a + bt - (v_a \cdot (c - p_a + bt) v_a))^2 - r^2 = 0$$

reduces to $At^2 + Bt + C = 0$

with

$$A = (b - (b \cdot v_a)v_a)^2$$

$$B = 2((b - (b \cdot v_a)v_a) \cdot (cp - (cp \cdot v_a)v_a))$$

$$C = (cp - (cp \cdot v_a)v_a)^2 - r^2$$

where $cp = c - p_a$

Cylinder caps

A finite cylinder with caps can be constructed as the intersection of an infinite cylinder with a slab between two parallel planes, which are perpendicular to the axis.

To intersect a ray with a cylinder with caps:

- **intersect with the infinite cylinder;**
- **check if the intersection is between the planes;**
- **intersect with each plane;**
- **determine if the intersections are inside caps;**
- **out of all intersections choose the one with minimal t**

Cylinder-ray intersections

POV -ray like cylinder with caps : cap centers at p_1 and p_2 , radius r .

Infinite cylinder equation: $p_a = p_1$, $v_a = (p_2 - p_1) / |p_2 - p_1|$

The finite cylinder (without caps) is described by equations:

$$(q - p_a - (v_a \cdot (q - p_a))v_a)^2 - r^2 = 0 \text{ and } (v_a \cdot (q - p_1)) > 0$$

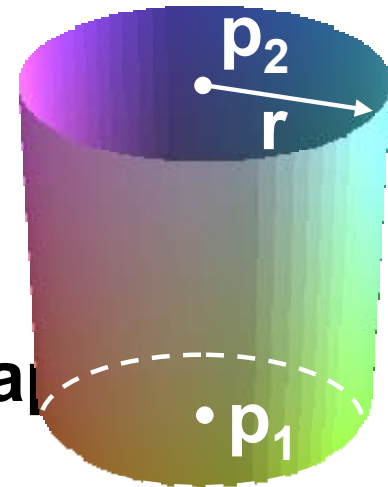
and

$$(v_a \cdot (q - p_2)) < 0$$

The equations for caps are:

$$(v_a \cdot (q - p_1)) = 0, (q - p_1)^2 < r^2 \quad \text{bottom cap}$$

$$(v_a \cdot (q - p_2)) = 0, (q - p_2)^2 < r^2 \quad \text{top cap}$$



Cylinder-ray intersections

Algorithm with equations:

Step 1: Find solutions t_1 and t_2 of $At^2 + Bt + C = 0$

if they exist. Mark as intersection candidates the one(s) that are nonnegative and for which $(v_a \cdot (q_i - p_1)) > 0$ and $(v_a \cdot (q_i - p_2)) < 0$, where $q_i = p + v t_i$

Step 2: Compute t_3 and t_4 , the parameter values for which the ray intersects the upper and lower planes of the caps.

If these intersections exist, mark as intersection candidates those that are nonnegative and $(q_3 - p_1)^2 < r^2$ (respectively $(q_4 - p_2)^2 < r^2$).

In the set of candidates, pick the one with min. t .

General quadrics

A general quadric has equation

$$Ax^2 + By^2 + Cz^2 + Dxy + Eyz + Fxz + Gx + Hy + Iz + J = 0$$

Intersections with general quadrics are computed in a similar way: For ray $c+bt$,

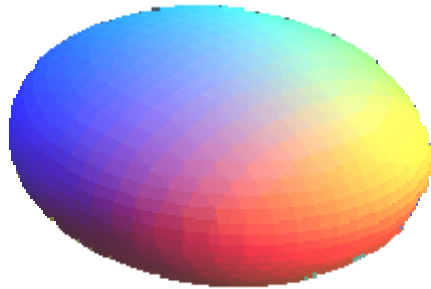
$$\text{take } x = c^x + b^x t, y = c^y + b^y t, z = c^z + b^z t$$

and solve the equation for t ; if there are solutions, take the smaller nonnegative one.

Infinite cones and cylinders are special cases of general quadrics.

General quadrics

Nondegenerate quadrics



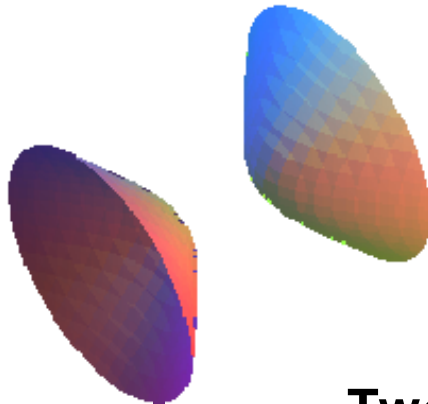
Ellipsoid

$$x^2/a^2 + y^2/b^2 + z^2/c^2 + 1 = 0$$



One-sheet hyperboloid

$$x^2/a^2 - y^2/b^2 + z^2/c^2 + 1 = 0$$

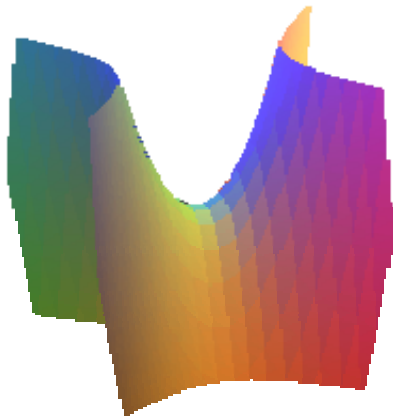


Two-sheet hyperboloid

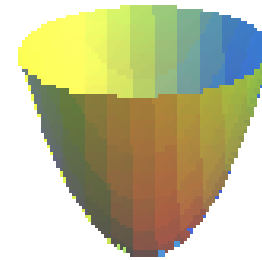
$$x^2/a^2 - y^2/b^2 + z^2/c^2 - 1 = 0$$

General quadrics

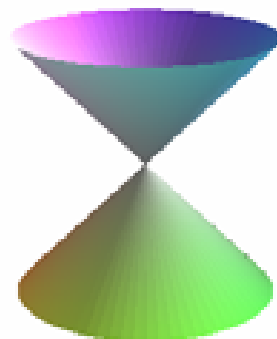
Nondegenerate quadrics



Hyperbolic paraboloid
 $x^2/a^2 - z^2/c^2 - 2y = 0$



Elliptic paraboloid
 $x^2/a^2 + z^2/c^2 - 2y = 0$



cone
 $x^2/a^2 - y^2/b^2 + z^2/c^2 = 0$

General quadrics

Degenerate quadrics

- planes (no quadratic terms),
- pairs of parallel planes (e.g. $x^2 - 1 = 0$)
- pairs of intersecting planes (e.g. $(x-1)^2 = 0$)
- elliptic cylinders (e.g. $x^2+z^2-1=0$)
- hyperbolic cylinders (e.g. $x^2-z^2-1=0$)
- parabolic cylinders (e.g. $x^2 -z = 0$)

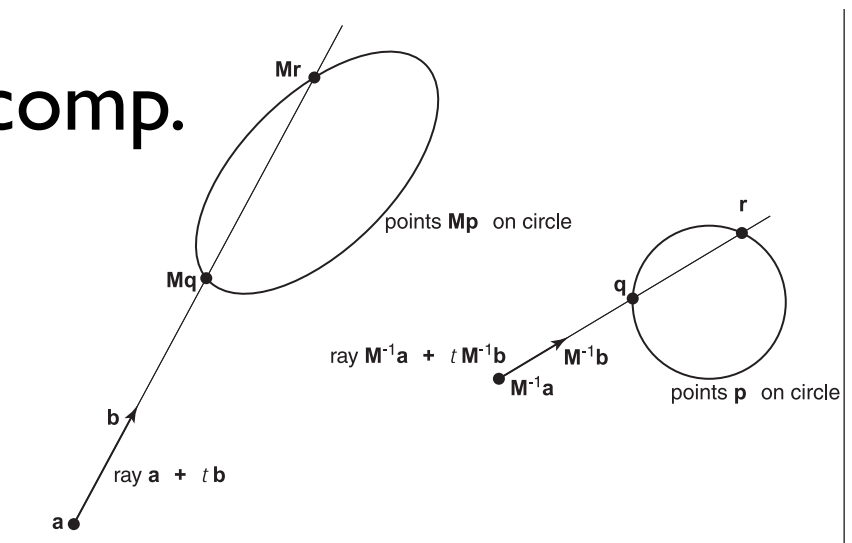
Possible to get “imaginary” surfaces (that is, with no points)! Example: $x^2 + 1 = 0$

Transformations

- Two options:
 - Apply transformation to the object and then intersect with ray
 - Apply the inverse of the transformation to the ray and then intersect with untransformed object

- easier intersection comp.

- instances



Basic Algorithm

For each pixel {

 Shoot a ray from camera to pixel

 for all objects in scene

 Compute intersection with ray

Find object with closest intersection

 Display color using object + light property

}

Closest Intersection

- Each object intersection routine should return a 't' value.
- One should keep track of the smallest non-negative 't' entry among all objects.
- By the time all objects are processed we should know the object id and the position of point on primitive.

Basic Algorithm

For each pixel {

 Shoot a ray from camera to pixel

 for all objects in scene

 Compute intersection with ray

 Find object with closest intersection

 Display color using object + light property

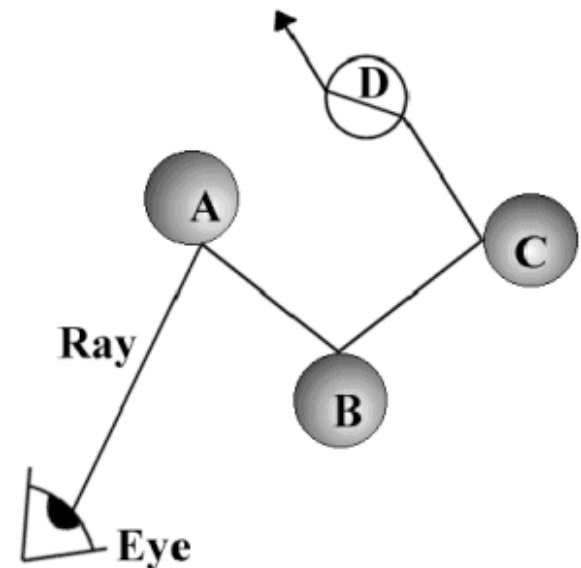
}

Color

- Color - Light and Material Properties
- For each pixel - compute value for each color component separately
- if no light - flat shading: pixel gets color of object
- if there's light - many more things to consider...
- solid/image texture

Ray Casting vs. Ray Tracing

- Ray Casting only finds visible surfaces given a point of view.
- Ray Tracing allows for more rays to be shot from the visible surface point to compute lighting effects ; shadows, reflection, refraction
- Ray Tracing can be defined as a recursion ... a TREE!

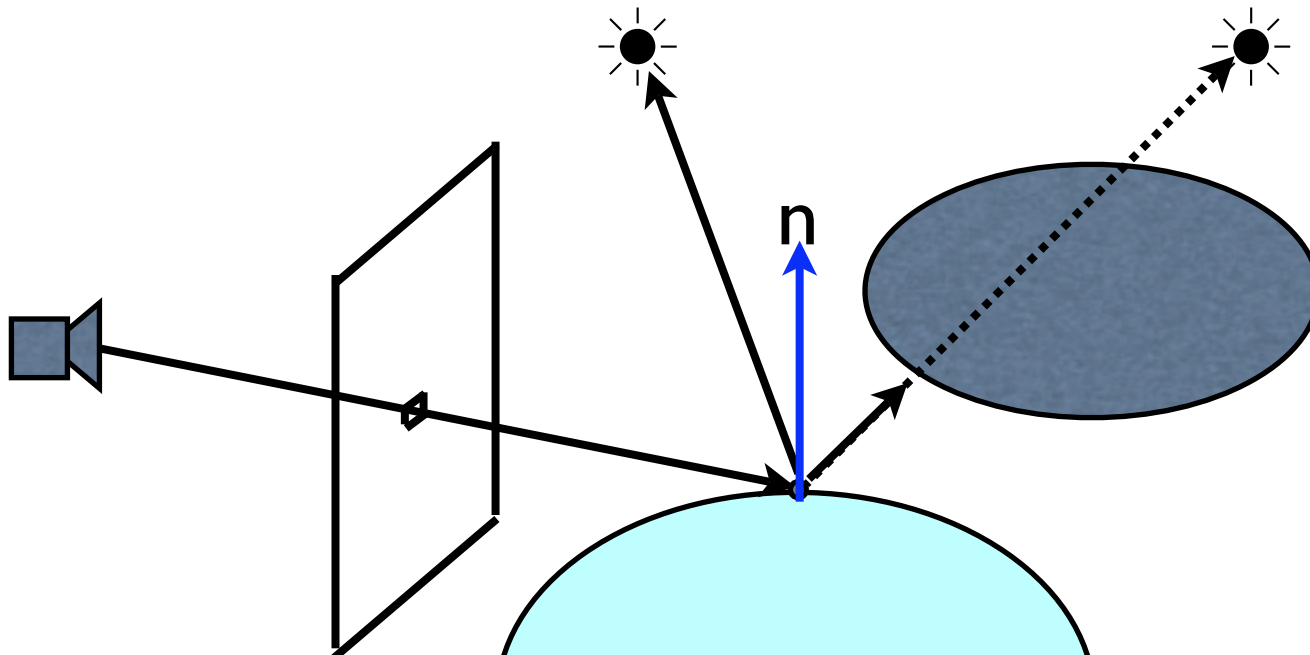


Lighting and Shadows

- One needs to include contribution from all light sources
- How to do with ray tracing:
 - Shoot rays to all light sources!

Lighting and Shadows

- We have : the point of intersection.
- We can compute normals using object information + transformation.
- From that point - shoot (shadow) rays to each (point) light source



Shadow Algorithm

At intersection point p

for all light sources s :

Shoot a ray from p_ϵ to s .

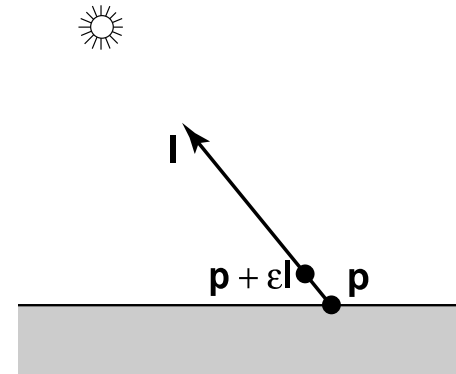
If the ray hits an object

No need to compute closest intersection, position etc.

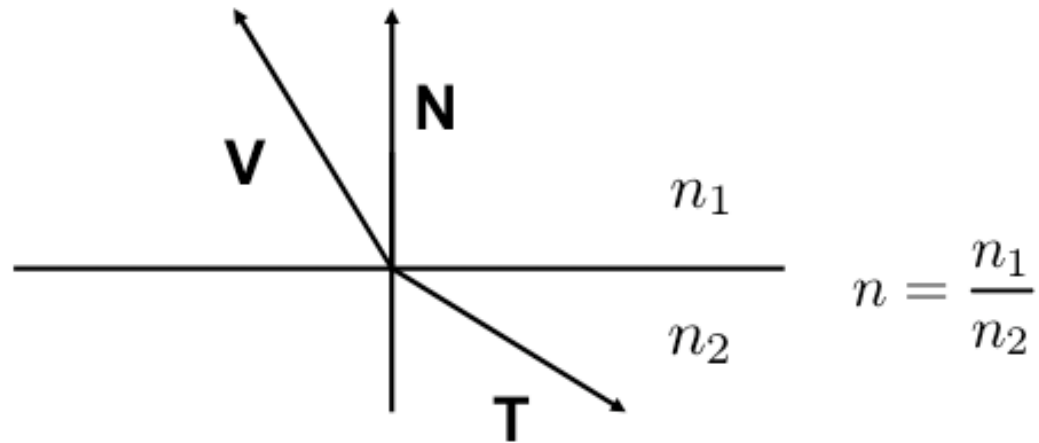
in shadow i.e. no contribution from that light

else

add contribution from light using normal and light/object properties

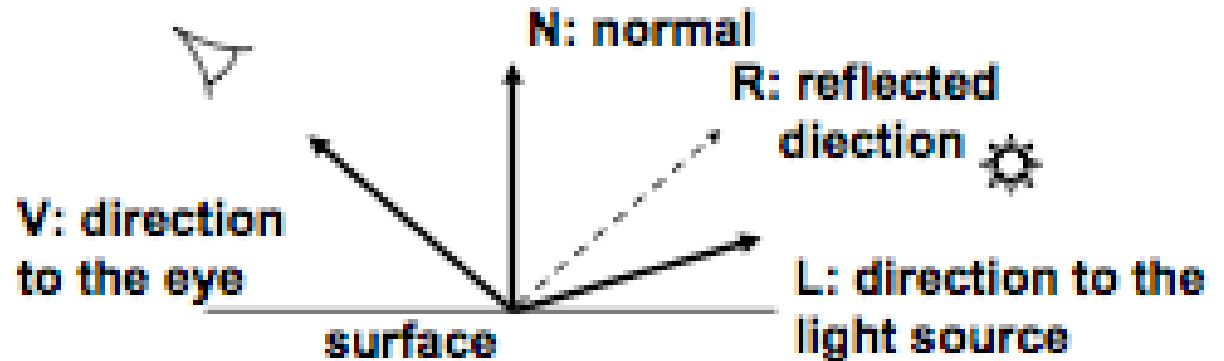


Refraction (for dielectrics)



$$\mathbf{T} = -n\mathbf{V} + \left(n(\mathbf{V} \cdot \mathbf{N}) - \sqrt{1 - n^2(1 - (\mathbf{V} \cdot \mathbf{N})^2)} \right) \mathbf{N}$$

Reflection



$$\mathbf{R} = 2(\mathbf{N}, \mathbf{V})\mathbf{N} - \mathbf{V}$$

Plus, the change in sign in V!!

Ray tracing

For each pixel shoot a ray R from camera;
`pixel = TraceRay(R)`

The recursive ray tracing procedure:

```
RGBvalue TraceRay(Ray R)
  shoot rays to all light sources;
  for all visible sources, compute RGB values  $r_i$ 
  shoot reflected ray  $R_{refl}$ ;  $r_{refl} = \text{TraceRay}(R_{refl})$ 
  shoot refracted ray  $R_{trans}$ ;  $r_{trans} = \text{TraceRay}(R_{trans})$ 
  compute resulting RGB value from
   $r_i$ ,  $r_{refl}$ ,  $r_{trans}$  using the lighting model
```

One Lighting Model (From POV-Ray!)

- Given M visible light sources from a point with colors: $\langle r_i, g_i, b_i \rangle$ for $i=1:M$
- Object color $\langle m_r, m_g, m_b, m_f \rangle$
- And the lighting parameters:

```
ambient  $k_{amb}$   
diffuse  $k_{diff}$   
phong  $k_{spec}$   
phong_size  $p$   
reflection  $k_{refl}$ 
```

For red only:

$$\text{final color} = (1 - m_f)k_{amb}m_r r_{amb} + \sum_{i=1}^M (1 - m_f)r_i (k_{diff}m_r(N, L_i) + k_{spec}m_r(R, L_i)^p) + k_{refl}r_{refl} + m_fm_r r_{trans}$$

L_i : direction to light source i ,

N_i : normal,

R_i : reflected direction

Infinite trace?

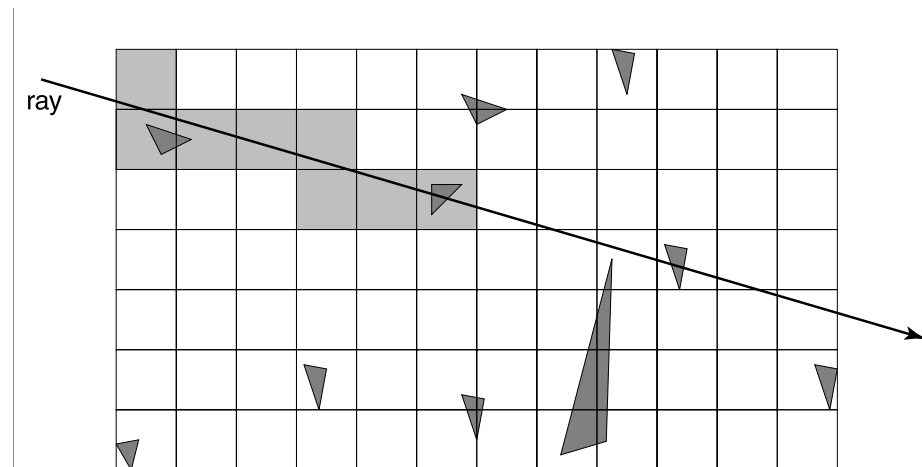
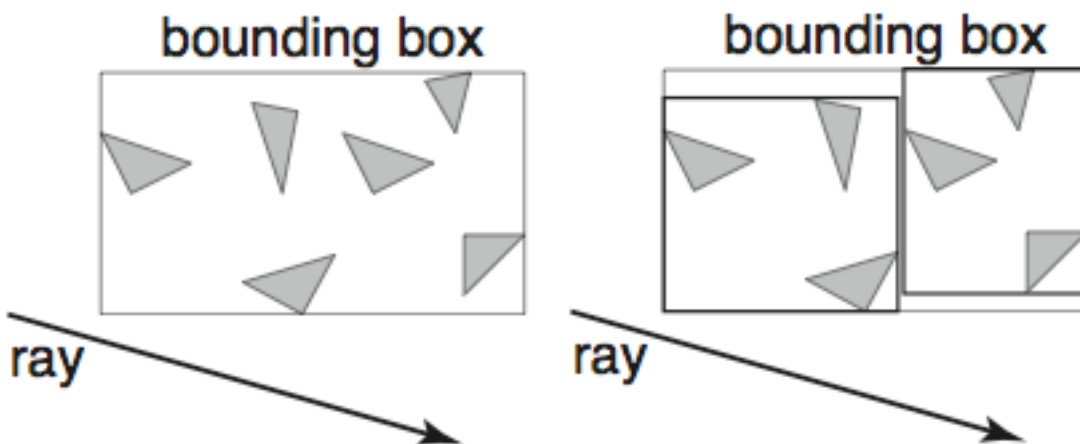
- maximum depth
- generate reflected ray only if specular color of object at intersection point is non-zero

Efficiency Issues

Approximately 95% of the time in ray tracing is spend on intersection tests

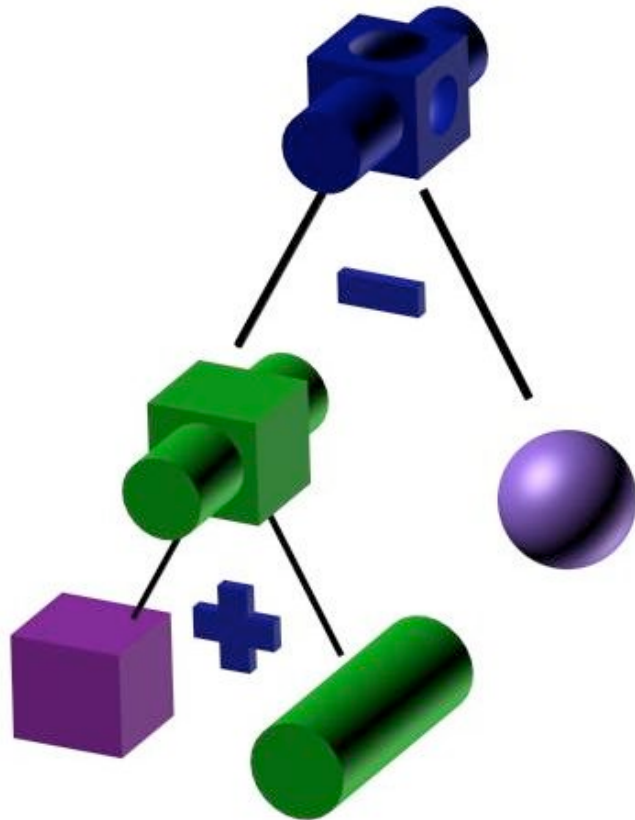
Three basic approaches for speed-up

- To compute fewer intersections : adaptive depth control, adaptive sampling
- To use generalized rays - beams, cones
- To compute intersections faster - bounding boxes (boolean), hierarchical bbs, spatial subdivision

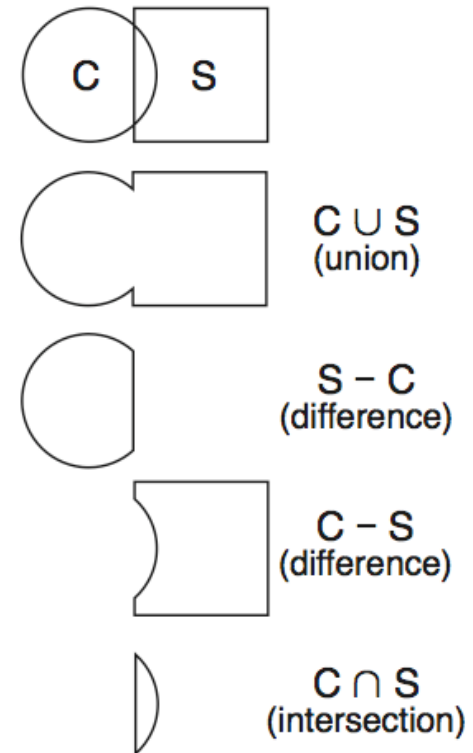


Constructive Solid Geometry

- CSG: Using set operations for solid shapes



http://escience.anu.edu.au/lecture/cg/surfaceModeling/CSG_tree.en.html

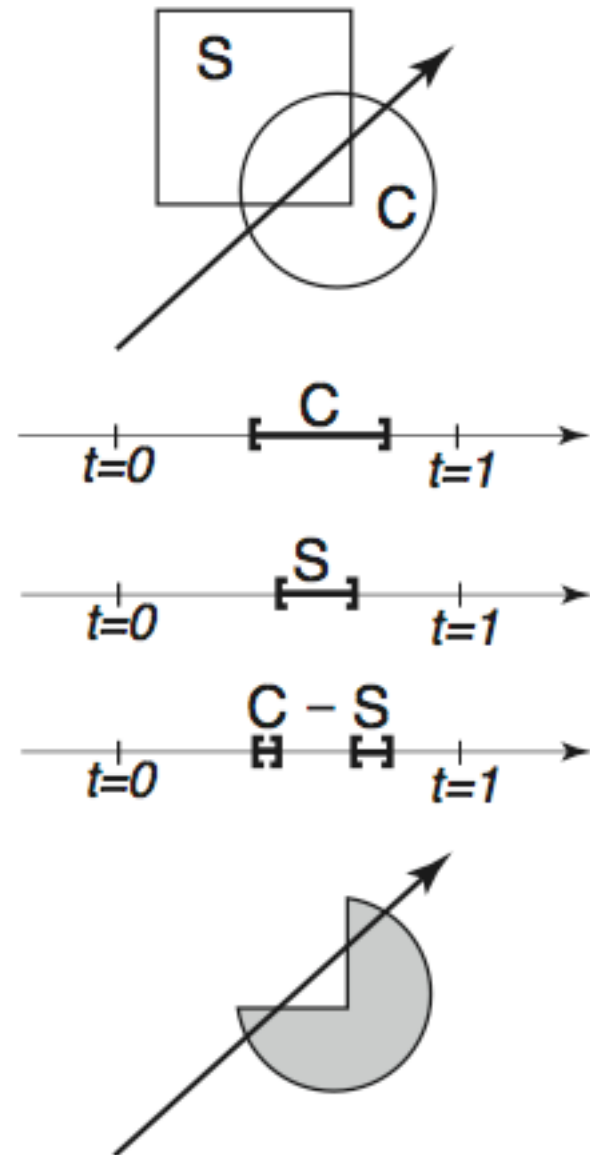


<http://www.cs.utah.edu/~shirley/fcg/figures/ray/>

- If only an image is needed - need not change the models

Constructive Solid Geometry

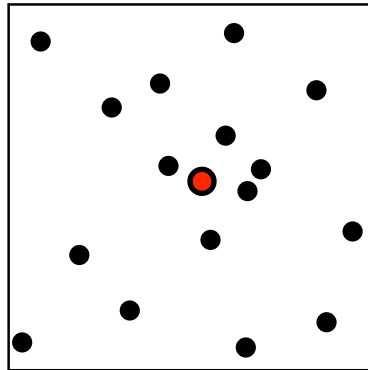
- Find all intersections with a model, instead of just the closest.
- find intervals where a ray is inside the object
- Do set operations on intervals



Other Effects

Distribution Ray Tracing

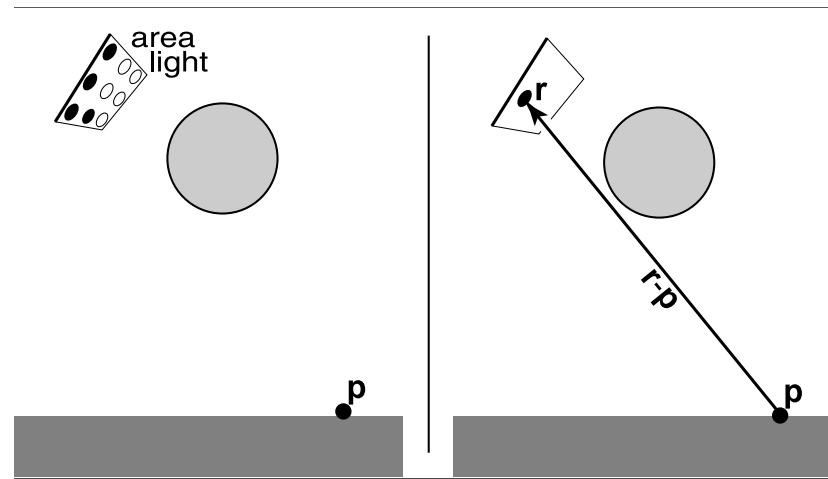
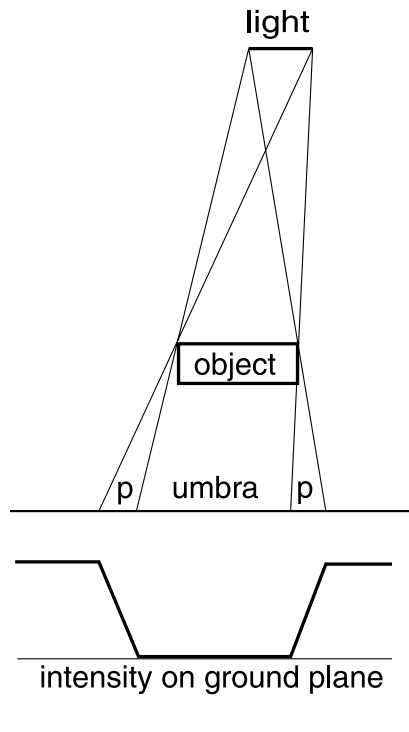
- Antialiasing - using samples within a pixel



Other Effects

Distribution Ray Tracing

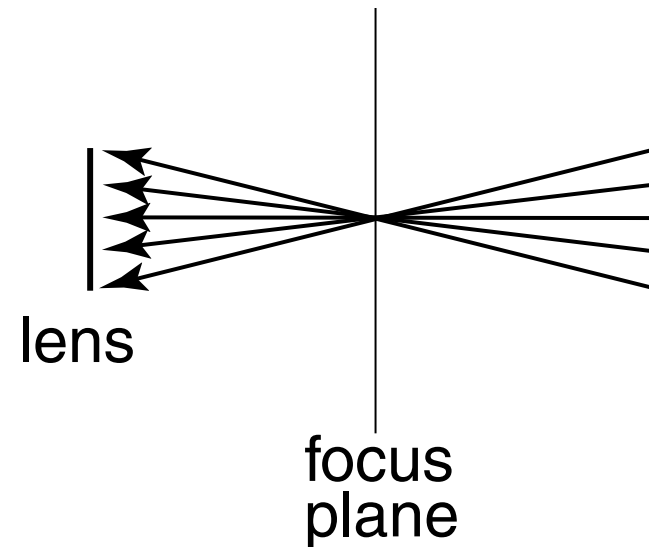
- Antialiasing - using samples within a pixel
- Soft Shadows



Other Effects

Distribution Ray Tracing

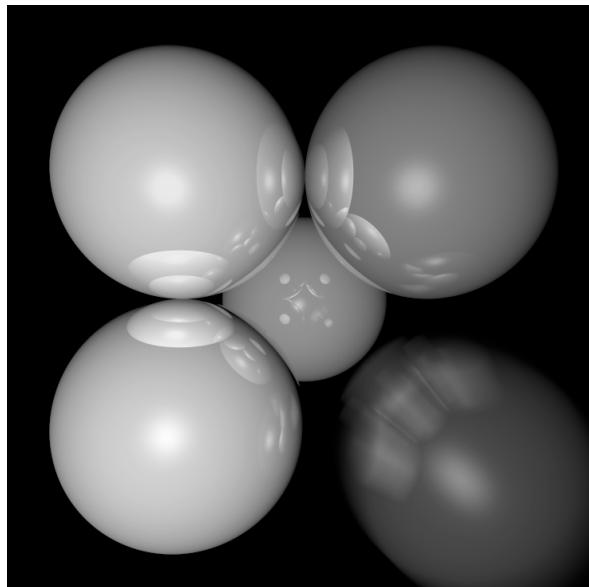
- Antialiasing - using samples within a pixel
- Soft Shadows
- Depth of Field



Other Effects

Distribution Ray Tracing

- Antialiasing - using samples within a pixel
- Soft Shadows
- Depth of Field
- Motion Blur - rays shot within a time interval



Hw Notes

Hw7

- Download and experiment with POV-Ray
- Starting point:
 - Create a base Shape class with virtual functions to compute intersections.
 - Derive each shape from this class
 - Implement sphere, polygon, cylinder, box, quadric with corresponding intersection functions.
 - easy to add bounding boxes, new shapes, etc.

Hw Notes : General Organization

- Two main parts : Rendering engine + modules to manage parameters (objects, camera, lights, etc)
- Use a base Shape Class with virtual functions to compute intersections. Derive each shape from this class
- Create a ray data structure: origin, direction, color
- Vector class
- Parser

Hw Notes: Transformations

Two Approaches:

- Can transform each object before rendering
- Transform the ray to object's coord system

When intersecting a ray with a transformed object, apply inverse of the transformation to the ray.

(i.e. multiply origin and direction of ray with inverse transformation matrix)

Can store the inverse of the transformation matrix with the object

Hw Notes: Ray Tracing

Note that your homework is far from a complete ray tracer. You need at least

- A Lighting Model based on object properties, including reflection, refraction
- Recursive ray tracing. (for hw we only use camera-object, object-light source)
- Possibly a speed up - e.g. bounding boxes

A simple lighting model is provided with the hw description, feel free to implement that or anything else from this list for extra credit.