Announcements

- Assignment 3 due today

  Questions???

- Remember that you have late days (if you haven't used them yet…)
- Problem set 3 out at the end of the day
- Movie for Assignment 2 at the end of class

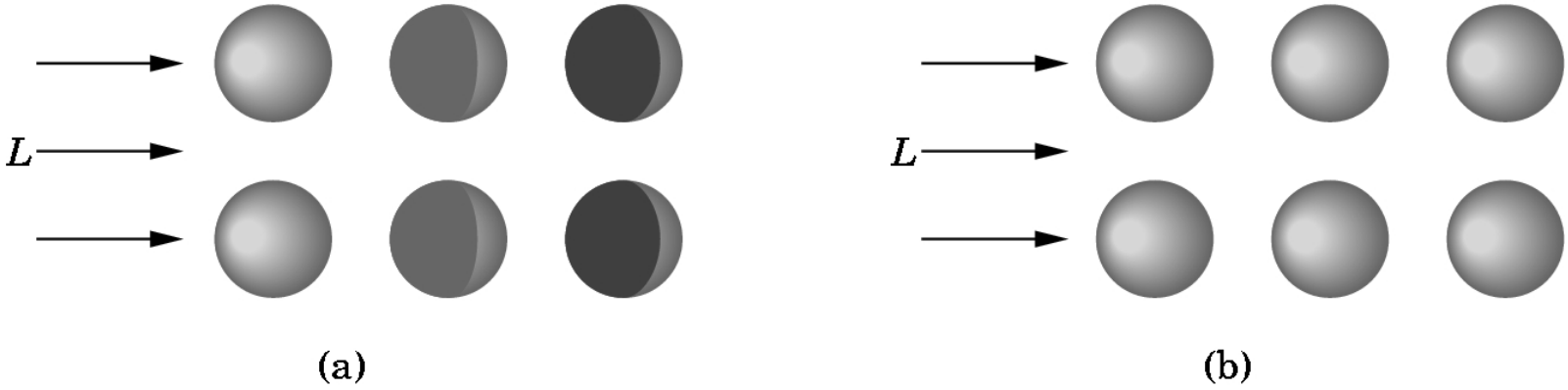
Ray Casting

Forward & Backward Ray Tracing
Ray Casting
Ray-Surface Intersection Testing
Barycentric Coordinates

Watt 1.4 and 12

Global vs. Local Rendering Models

- We've been talking about local rendering models. The color of one object is independent of its neighbors (except for shadows)
- Missing scattering of light between objects, real shadowing
- Global Rendering Models
  - Raytracing—specular highlights
  - Radiosity—diffuse surfaces, closed environments



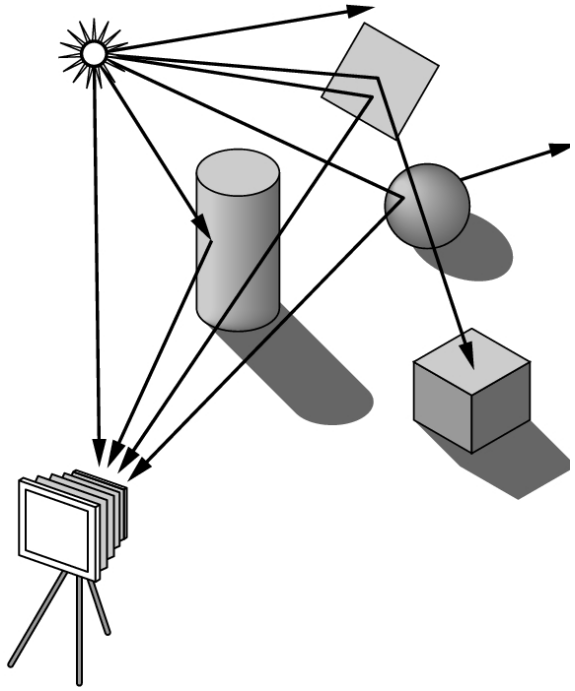
(a) (b)

Light is Bouncing Photons

- Light sources send off photons in all directions
  - Model these as particles that bounce off objects in the scene
  - Each photon has a wavelength and energy (color and intensity)
  - When photons bounce, some energy is absorbed, some reflected, some transmitted
- If we can model photon bounces we can generate images
- Technique: follow each photon from the light source until:
  - All of its energy is absorbed (after too many bounces)
  - It departs the known universe
  - It strikes the image and its contribution is added to appropriate pixel

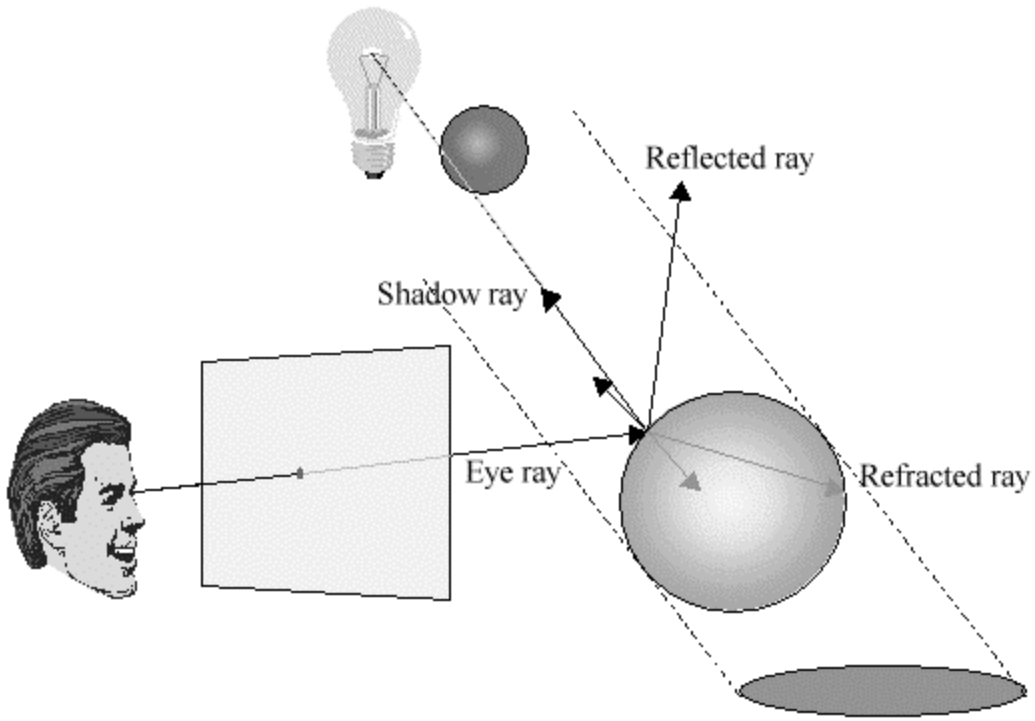
Forward Ray Tracing

- Rays are the paths of these photons
- This method of rendering by following photon paths is called *ray tracing*
- *Forward* ray tracing follows the photon in direction that light travels (from the source)
- BIG problem with this approach:
  - Only a tiny fraction of rays reach the image
  - Extremely slow
- Ideal Scenario:
  - We'd like to magically know which rays will eventually contribute to the image, and trace only those



Backward Ray Tracing

- The solution is to start from the image and trace backwards - *backward* ray tracing
  - Start from the image and follow the ray until the ray finds (or fails to find) a light source



Backward Ray Tracing

- Basic ideas:
  - Each pixel gets light from just one direction - the line through the image point and focal point
  - Any photon contributing to that pixel's color has to come from this direction
  - So head in that direction and find out what is sending light this way
  - If we hit a light source - we're done
  - If we find nothing - we're done
  - If we hit a surface - see where that surface is lit from
- At the end we've done forward ray tracing, but ONLY for the rays that contribute to the image

Ray Casting

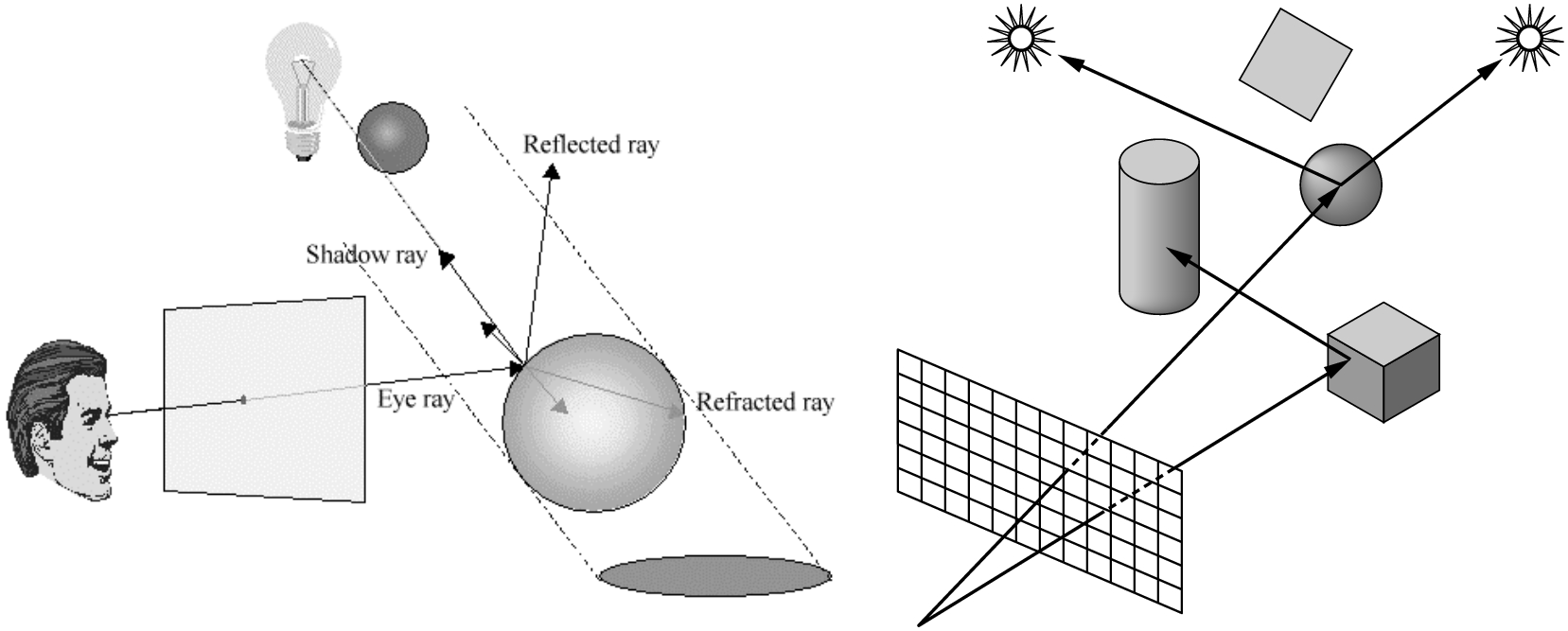
- This version of ray tracing is often called *ray casting*
- The algorithm is

```
loop y
    loop x
      shoot ray from eye point through pixel (x,y) into scene
      intersect with all surfaces, find first one the ray hits
      shade that point to compute pixel (x,y)'s color
        (perhaps simulating shadows as we discussed earlier)
```

- A ray is p+*t*d: p is ray origin, d the 3D direction
  - *t*=0 at origin of ray, *t*>0 in positive direction of ray
  - typically assume ||d||=1
  - p and d are typically computed in world space

- This is easily generalized to give recursive *ray tracing*...

Recursive Ray Tracing



- We'll distinguish four ray types:
  - Eye rays: originate at the eye
  - Shadow rays: from surface point toward light source
  - Reflection rays: from surface point in mirror direction
  - Transmission rays: from surface point in refracted direction
- Trace all of these recursively. *More on this later.*

Writing a Simple Ray Caster (no bounces)

```
Raycast()                  // generate a picture
   for each pixel x,y
      color(pixel) = Trace(ray_through_pixel(x,y))

Trace(ray)                 // fire a ray, return RGB radiance
                           // of light traveling backward along it
   object_point = Closest_intersection(ray)
   if object_point return Shade(object_point, ray)
   else return Background_Color

Closest_intersection(ray)
   for each surface in scene
        calc_intersection(ray, surface)
   return the closest point of intersection to viewer
   (also return other info about that point, e.g., surface
     normal, material properties, etc.)

Shade(point, ray)          // return radiance of light leaving
                           // point in opposite of ray direction
   calculate surface normal vector
   use Phong illumination formula (or something similar)
   to calculate contributions of each light source
```

Ray-Surface Intersections

- Ray equation: (given origin p and direction d)

  $x(t) = p + td$

- Surfaces can be represented by:
  - Implicit functions: $f(x) = 0$
  - Parametric functions: $x = g(u,v)$
- Compute Intersections:
  - Substitute ray equation for x
  - Find roots
  - Implicit: $f(p + td) = 0$
    - » one equation in one unknown – univariate root finding
  - Parametric: $p + td - g(u,v) = 0$
    - » three equations in three unknowns (t,u,v) – multivariate root finding
  - For univariate polynomials, use closed form solution otherwise use numerical root finder

The Devil's in the Details

- Solving these intersection equations can be tough...
  - General case: non-linear root finding problem
  - Simple surfaces can yield a closed-form solution
  - But generally a numerical root-finding method is required
    - » Expensive to calculate
    - » Won't always converge
    - » When repeated millions of times, special cases WILL occur
- The good news:
  - Ray tracing is simplified using object-oriented techniques
    - » Implement one intersection method for each type of surface primitive
    - » Each surface handles its own intersection
  - Some surfaces yield closed form solutions:
    - » quadrics: spheres, cylinders, cones, ellipsoids, etc…
    - » polygons
    - » tori, superquadrics, low-order spline surface patches

Ray-Sphere Intersection

- Ray-sphere intersection is an easy case
- A sphere's implicit function is $x^2+y^2+z^2-r^2=0$ if sphere at origin
- The ray equation is: $x = p_x+td_x$

  $y = p_y+td_y$

  $z = p_z+td_z$
- Substitution gives: $(p_x+td_x)^2 + (p_y+td_y)^2 + (p_z+td_z)^2 - r^2 = 0$
- A quadratic equation in *t*.
- Solve the standard way: $A = d_x^2+d_y^2+d_z^2 = 1$ (unit vec.)

  $At^2+Bt+C=0$

  $B = 2(p_xd_x+p_yd_y+p_zd_z)$

  $C = p_x^2+p_y^2+p_z^2 - r^2$

Ray-Sphere Intersection continued

Quadratic formula has two roots: $t=(-B\pm sqrt(B^2-4C))/2$

- Real roots correspond to the two intersection points
- Negative determinant means ray misses sphere $(B^2 - 4C < 0)$
- Determinant = 0 means ray grazes sphere

We also need the normal, for sphere centered at (l, m, n)

$$N = \left(\frac{x_i - l}{r}, \frac{y_i - m}{r}, \frac{z_i - n}{r} \right)$$

Ray-Polygon Intersection

Assuming we have a planar polygon

- first, find intersection point of ray with plane that contains polygon
- then check if that point is inside the polygon

- Intersection of ray with polygon the easy way (faster way in a minute):

$$ax + by + cy + d = 0$$

$$x = x_1 + it$$

$$y = y_1 + jt$$

$$z = z_1 + kt$$

$$t = -\frac{ax_1 + by_1 + cz_1 + d}{ai + bj + ck}$$

$ai + bj + ck = 0$ (ray/plane parallel

Ray-Polygon Intersection

Assuming we have a planar polygon

- first, find intersection point of ray with plane that contains polygon
- then check if that point is inside the polygon

- Latter step is a point-in-polygon test in 3-D:
  - inputs: a point x in 3-D and the vertices of a polygon in 3-D
  - output: INSIDE or OUTSIDE
  - problem can be reduced to point-in-polygon test in 2-D
- Point-in-polygon test in 2-D:
  - easiest for triangles
  - easy for convex n-gons
  - harder for concave polygons
  - most common approach: subdivide all polygons into triangles
  - for optimization tips, see article by Haines in the book *Graphics Gems IV*

Ray-Plane Intersection—again

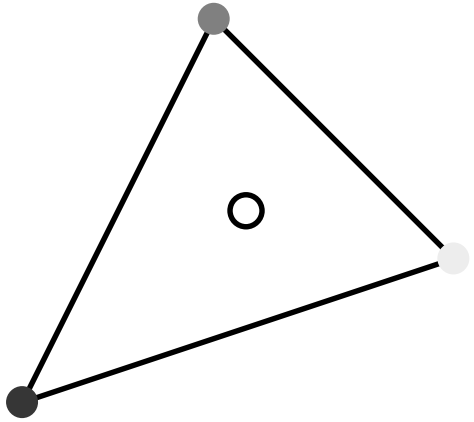
- Ray: x=p+*t*d
  - where p is ray origin, d is ray direction. we'll assume ||d||=1 (this simplifies the algebra later)
  - x(*t*)=*(x,y,z)* is point on ray if *t*>0
- Plane: (x-q)•n=0
  - where q is reference point on plane, n is plane normal. (some assume ||n||=1; we won't)
  - x is point on plane
  - if what you're given is vertices of a polygon
    - » compute n with cross product of two (non-parallel) edges
    - » use one of the vertices for q
  - rewrite plane equation as x•n+*D=0*
    - » equivalent to the familiar formula *Ax+By+Cz+D=0*, where (*A,B,C*)=n, *D*=-q•n
    - » fewer values to store
- Steps:
  - substitute ray formula into plane equation, yielding 1 equation in 1 unknown (*t*).
  - solution: *t* = -(p•n+*D*)/(d•n)
    - » note: if d•n=0 then ray and plane are parallel - REJECT
    - » note: if *t*<0 then intersection with plane is behind ray origin - REJECT
  - compute *t*, plug it into ray equation to compute point x on plane

Projecting A Polygon from 3-D to 2-D

- Point-in-polygon testing is simpler and faster if we do it in 2-D
  - The simplest projections to compute are to the *xy*, *yz*, or *zx* planes
  - If the polygon has plane equation $Ax+By+Cz+D=0$, then
    - » $|A|$ is proportional to projection of polygon in *yz* plane
    - » $|B|$ is proportional to projection of polygon in *zx* plane
    - » $|C|$ is proportional to projection of polygon in *xy* plane
    - » Example: the plane $z=3$ has $(A,B,C,D)=(0,0,1,-3)$, so $|C|$ is the largest and *xy* projection is best. We should do point-in-polygon testing using *x* and *y* coords.
  - In other words, project into the plane for which the perpendicular component of the normal vector n is largest to maintain accuracy
- Optimization:
  - We should optimize the inner loop (ray-triangle intersection testing) as much as possible
  - We can determine which plane to project to, for each triangle, as a preprocess

Digression before we get to point-in-polygon testing: Interpolated Shading for Ray Tracing

- Suppose we know colors or normals at vertices
  - How do we compute the color/normal of a specified point inside?



- Color depends on distance to each vertex
  - Want this to be linear (so we get same answer as scanline algorithm such as Gouraud or Phong shading)
  - But how to do linear interpolation between 3 points?
  - Answer: *barycentric coordinates*
- Useful for ray-triangle intersection testing too!

Barycentric Coordinates in 1-D

- Linear interpolation between colors C_0 and C_1 by t

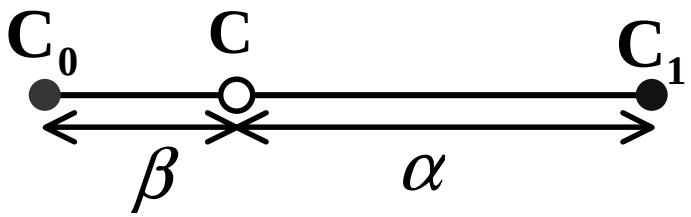
$$\mathbf{C} = (\mathbf{1} - t)\mathbf{C_0} + t\mathbf{C_1}$$

- We can rewrite this as

$$\mathbf{C} = \alpha\mathbf{C_0} + \beta\mathbf{C_1} \quad where\ \alpha + \beta = 1$$

$$\mathbf{C}\ \text{is between}\ \mathbf{C_0}\ \text{and}\ \mathbf{C_1} \Leftrightarrow \alpha, \beta \in [0,1]$$

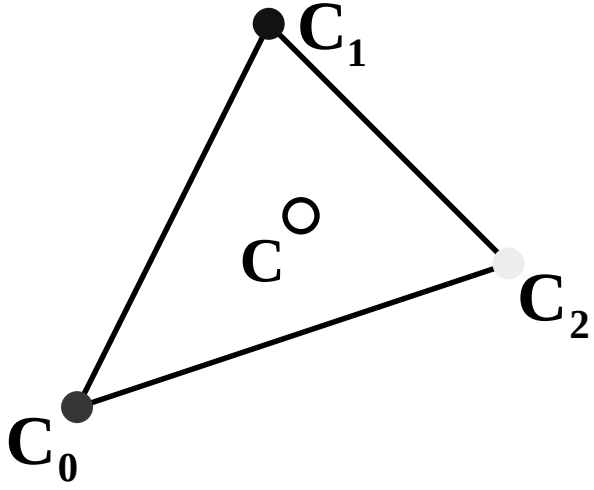
- Geometric intuition:
  - We are weighting each vertex by ratio of distances (or areas in the 2d case)



- α and β are called *barycentric* coordinates

Barycentric Coordinates in 2-D

- Now suppose we have 3 points instead of 2



- Define three barycentric coordinates: α, β, γ

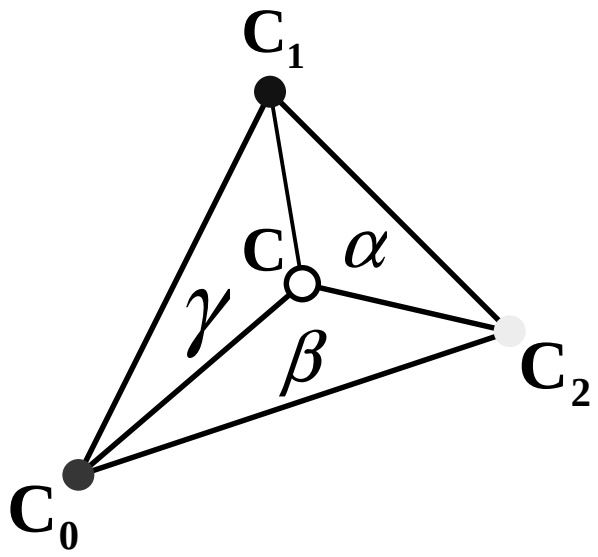
$$\mathbf{C} = \alpha\mathbf{C}_0 + \beta\mathbf{C}_1 + \gamma\mathbf{C}_2 \quad \text{where } \alpha + \beta + \gamma = 1$$

$$\mathbf{C} \text{ is inside } \mathbf{C}_0\mathbf{C}_1\mathbf{C}_2 \Leftrightarrow \alpha, \beta, \gamma \in [0,1]$$

- How to define α, β, and γ?

Barycentric Coordinates for a Triangle

- Define barycentric coordinates to be ratios of triangle areas



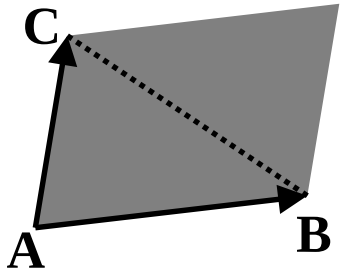
$$\alpha = \frac{Area(\mathbf{C}\mathbf{C}_1\mathbf{C}_2)}{Area(\mathbf{C}_0\mathbf{C}_1\mathbf{C}_2)}$$

$$\beta = \frac{Area(\mathbf{C}_0\mathbf{C}\mathbf{C}_2)}{Area(\mathbf{C}_0\mathbf{C}_1\mathbf{C}_2)}$$

$$\gamma = \frac{Area(\mathbf{C}_0\mathbf{C}_1\mathbf{C})}{Area(\mathbf{C}_0\mathbf{C}_1\mathbf{C}_2)} = 1 - \alpha - \beta$$

Computing Area of a Triangle

- in 3-D



  - *Area*(ABC) = parallelogram area / 2 = ||(B-A) x (C-A)||/*2*
  - faster: project to *xy*, *yz*, or *zx*, use 2D formula

- in 2-D
  - *Area*(*xy-projection*(ABC)) = $[(b_x-a_x)(c_y-a_y) - (c_x-a_x)(b_y-a_y)]/2$
    project A,B,C to *xy* plane, take *z* component of cross product
  - positive if ABC is CCW (counterclockwise)
  - Explained on next slide…

Computing Area of a Triangle - Algebra

That short formula,

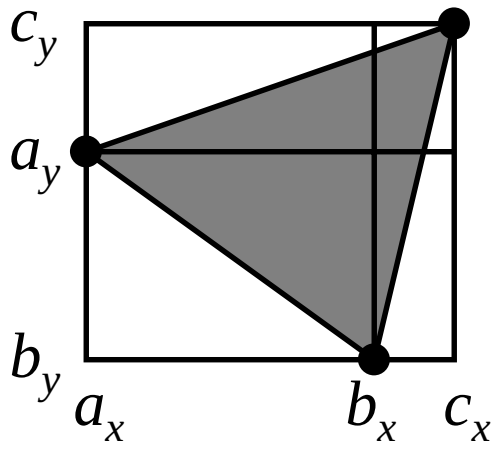
Area(ABC) = $[(b_x-a_x)(c_y-a_y) - (c_x-a_x)(b_y-a_y)]/2$

Where did it come from?

$$Area(ABC) = \frac{1}{2}\begin{vmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ 1 & 1 & 1 \end{vmatrix}$$

$$= \left(\begin{vmatrix} b_x & c_x \\ b_y & c_y \end{vmatrix} + \begin{vmatrix} c_x & a_x \\ c_y & a_y \end{vmatrix} + \begin{vmatrix} a_x & b_x \\ a_y & b_y \end{vmatrix} \right) / 2$$

$$= (b_x c_y - c_x b_y + c_x a_y - a_x c_y + a_x b_y - b_x a_y)/2$$



The short & long formulas above agree.

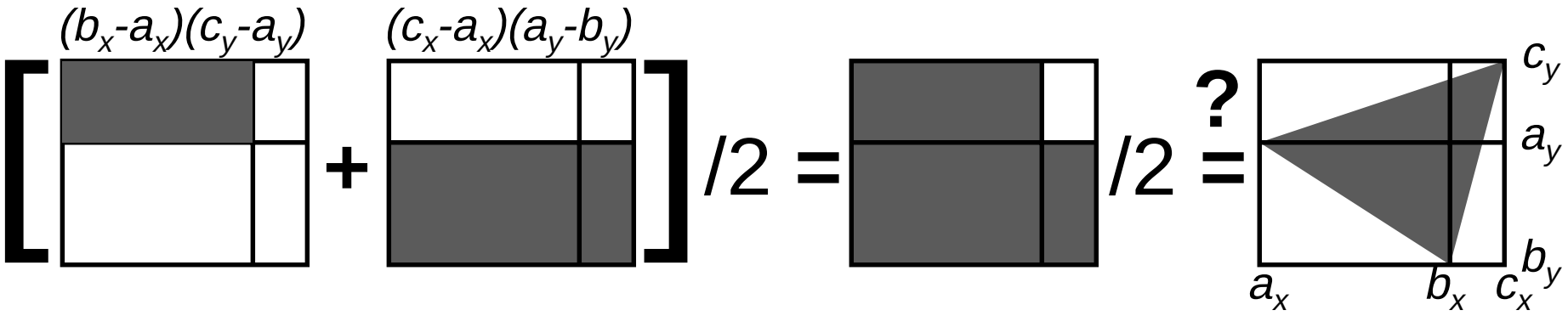
Short formula better because fewer multiplies. Speed is important!

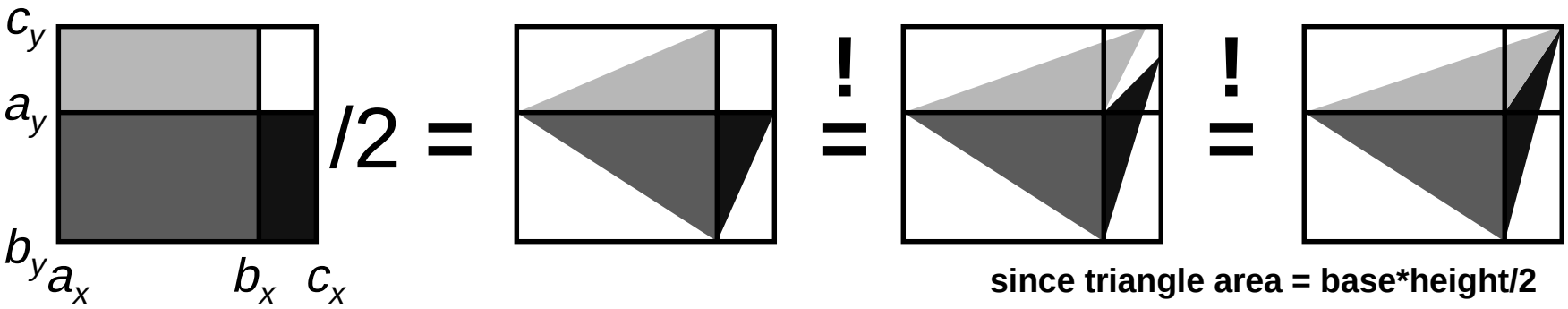
Can we explain the formulas geometrically?

Computing Area of a Triangle - Geometry

Area(ABC) $= [(b_x - a_x)(c_y - a_y) - (c_x - a_x)(b_y - a_y)]/2$

is a sum of rectangle areas, divided by 2.



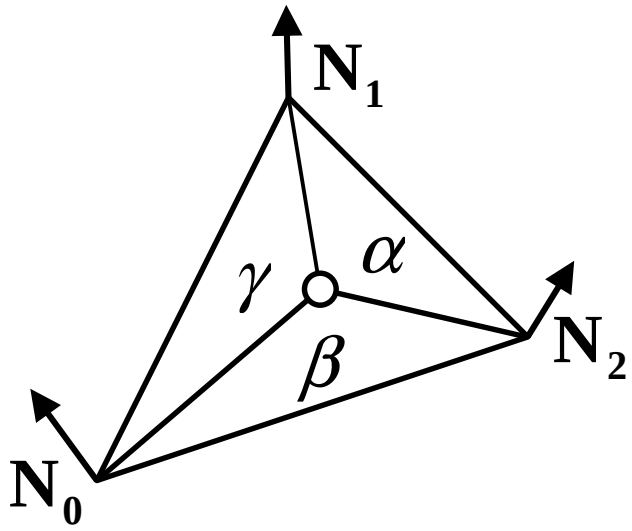


since triangle area = base*height/2

it works!

Uses for Barycentric Coordinates

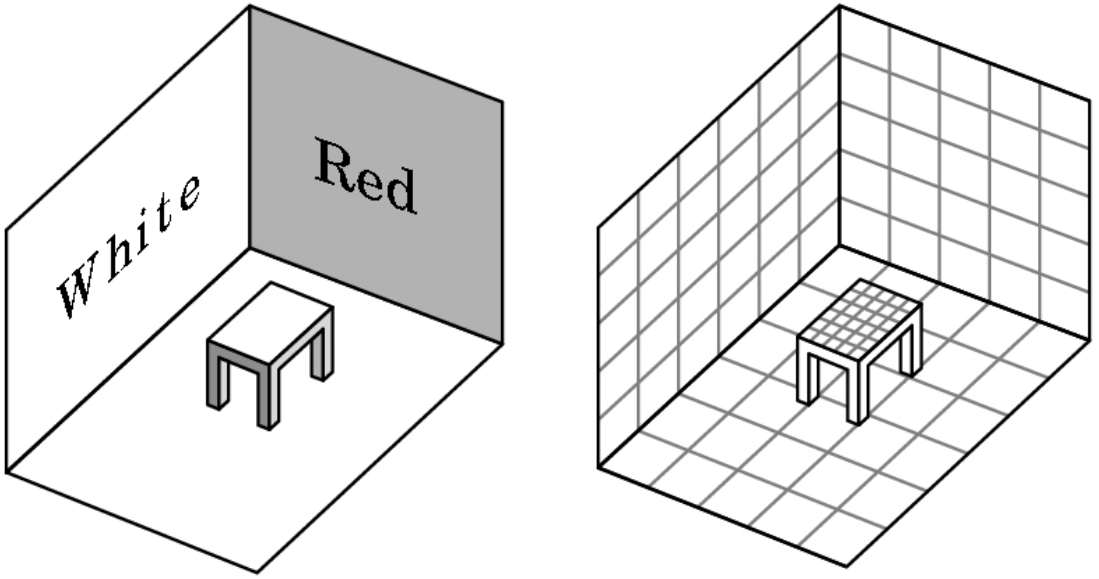
- Point-in-triangle testing!
  - point is in triangle iff $\alpha, \beta, \gamma > 0$
  - *note similarity to standard point-in-polygon methods that use tests of form $a_i x+b_i y+c_i<0$ for each edge i*



- Can use barycentric coordinates to interpolate *any* quantity
  - Gouraud Shading (color interpolation)
  - Phong Shading (normal interpolation)
  - Texture mapping ((*s,t*) texture coordinate interpolation)

Radiosity

We'll return to ray tracing in the next lecture but for the moment, here's a digression on another global rendering method.



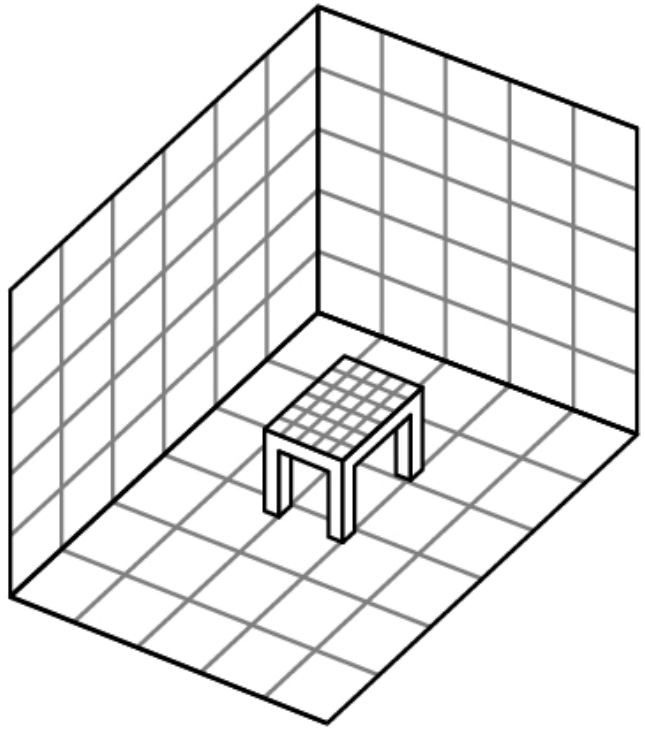
Simple scene with diffuse surfaces
White wall should show effect of being near red wall
Compute light reflected between each pair of patches

Radiosity Examples

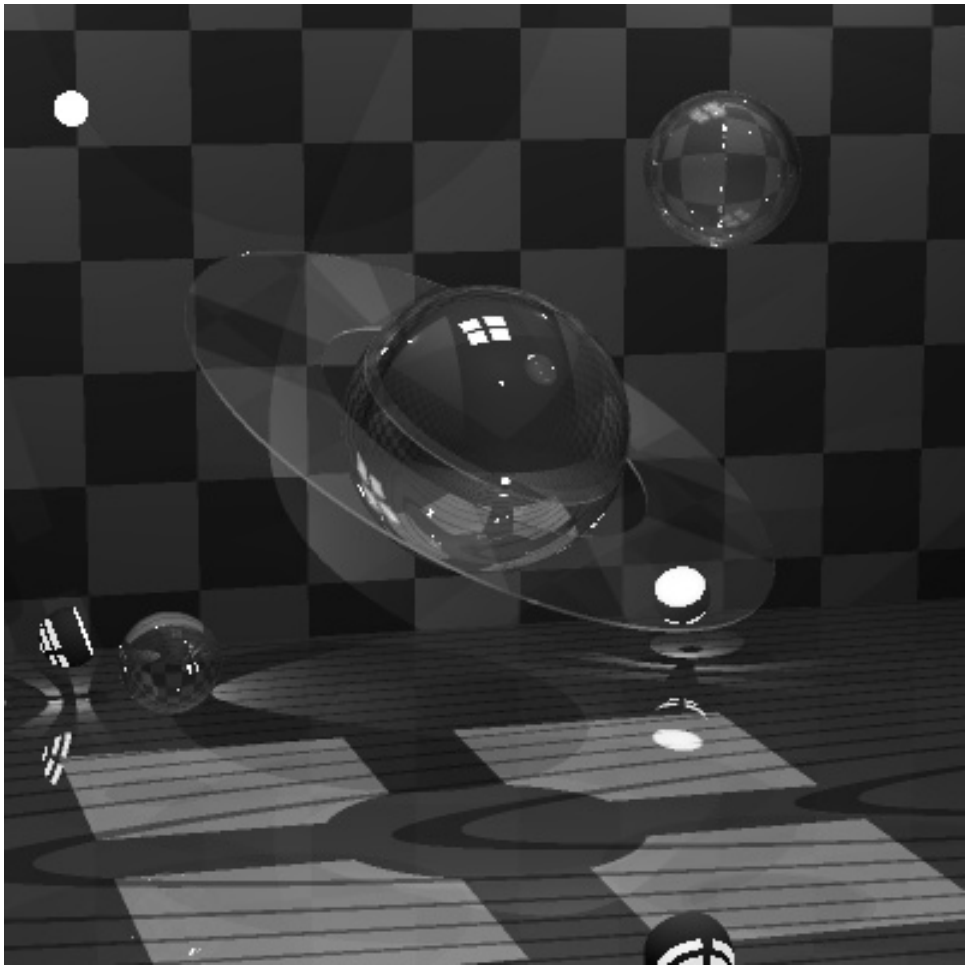
http://www.autodesk.com/us/lightscape/examples/html/index.htm

Radiosity

- Closed environment (office, factory)
- Compute interaction between all patches (over which intensity is assumed to be constant)
- View independent
- Difficult to do specular highlights
- Most impressive images to date



Raytracing Examples



http://www.med.osaka-u.ac.jp/pub/cl-comp/saito/raytr/saturn-img.html

Raytracing Examples

http://www.povray.org/

Raytracing Examples

http://www.povray.org/

Raytracing Examples

http://www.povray.org/

Radiosity Examples

http://www.autodesk.com/us/lightscape/examples/html/index.htm

Radiosity Examples

http://www.autodesk.com/us/lightscape/examples/html/index.htm

Radiosity Examples

http://www.autodesk.com/us/lightscape/examples/html/index.htm

Announcements

- Assignment 3 due today

  Questions???

- Remember that you have late days (if you haven't used them yet…)
- Problem set 3 out at the end of the day
- Movie for Assignment 2 at the end of class


Alkas Baybas