CS4160

COMPUTER GRAPHICS

class 5

today's class

code review: raytra 1.2 additions

more raytracing:

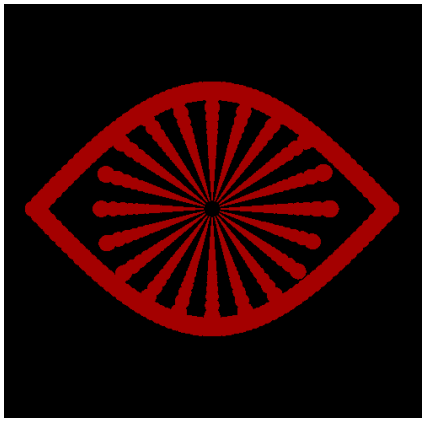
- triangles
- "true" shadows
- multiple light sources

animation appreciation:

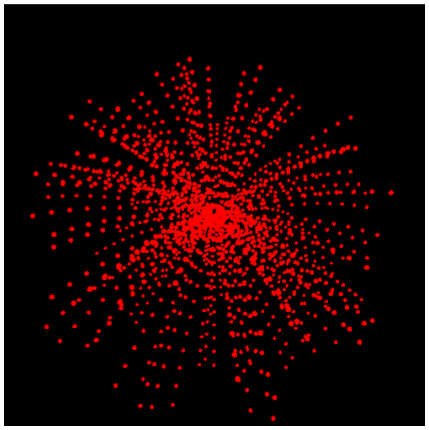
- "Knick Knack"

creative image submission: raytra 1.1

creative image submission: raytra 1.1
Eric Maciel - 3rd place



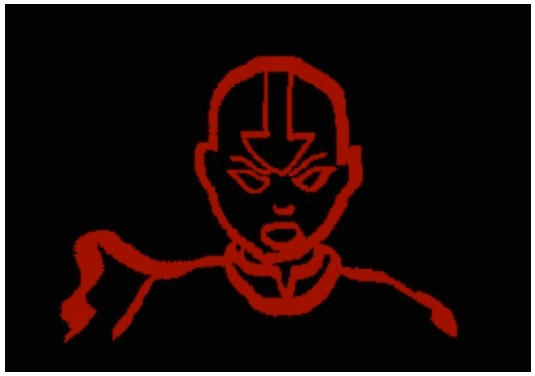
creative image submission: raytra 1.1
Tsung-Jui Tsai - 2nd place



5

5

creative image submission: raytra 1.1
Noah Zweben - 1st place!



6

6

recursive ray tracing - pseudocode

for each pixel:

- compute the ray through the current pixel (called a “primary ray”)
- intersect ray with the scene
- trace shadow rays to all lights
- compute shading at the intersection point
- if the surface is reflective, trace a reflection ray
- if the surface is transparent, trace a transmission ray
- combine shading, reflectance, transmission contributions into pixel value
- (if ray missed all objects, set color to background value)

7

recursive ray tracing - pseudocode

for each pixel:

- compute the ray through the current pixel (called a “primary ray”)
- intersect ray with the scene
- trace shadow rays to all lights
- compute shading at the intersection point
- if the surface is reflective, trace a reflection ray
- if the surface is transparent, trace a transmission ray
- combine shading, reflectance, transmission contributions into pixel value
- (if ray missed all objects, set color to background value)

8

triangles

triangles are the most common representation of geometry in 3D graphics:

- they can be used to represent, by planar elements, any shape to a desired error
- they fit well into ray tracing and pipeline paradigms
- they are easy to subdivide into smaller triangles (“level-of-detail”)
- most freeform geometry types (e.g. NURBS, subdivision surfaces) have well-defined methods of tesselation by triangles
- they are fast to compute intersections with
- disadvantages: only approximate smooth surfaces, without extra work normals are not smooth, all the other problems associated w/ sampling

simple triangle class

(note: this is the naive implementation:)

```
class tri  {
        // these points are in counterclockwise ordering
        // with respect to the normal.
        point p1;
        point p2;
        point p3;
        // optional, but will save recalc time as the
        // normal is needed on lighting computations
        vector nrml;
};
```


triangles - intersection

there are many, many ways to compute the intersection of a ray with a triangle.

here is one:

- compute intersection of ray with triangle’s plane
- check to see if intersection point is inside triangle

Ray-triangle intersection

- condition 1: point is on ray
  $$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$
- condition 2: point is on plane (**a** is any point on plane)
  $$(\mathbf{x} - \mathbf{a}) \cdot \mathbf{n} = 0$$
- condition 3: point is on the inside of all three edges

Ray-triangle intersection

- condition 1: point is on ray

$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$

- condition 2: point is on plane (**a** is any point on plane)

$$(\mathbf{x} - \mathbf{a}) \cdot \mathbf{n} = 0$$

- solve 1&2 (ray–plane intersection)
  – substitute and solve for *t*:

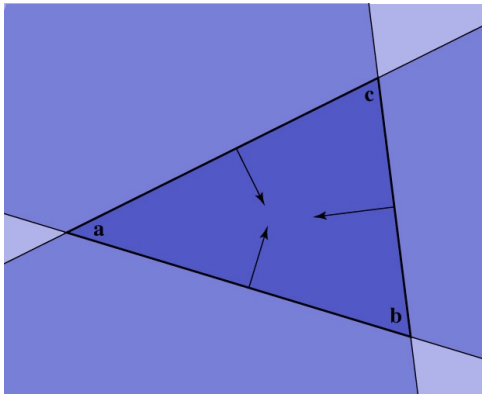
$$(\mathbf{p} + t\mathbf{d} - \mathbf{a}) \cdot \mathbf{n} = 0$$

$$t = \frac{(\mathbf{a} - \mathbf{p}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}$$

13

Ray-triangle intersection

- In plane, triangle is the intersection of 3 half spaces



14

Inside-edge test

- Need outside vs. inside
- Reduce to clockwise vs. counterclockwise
  – vector of edge to vector to **x**
- Use cross product to decide: is point **x** between vectors (defined by edges) at every vertex?

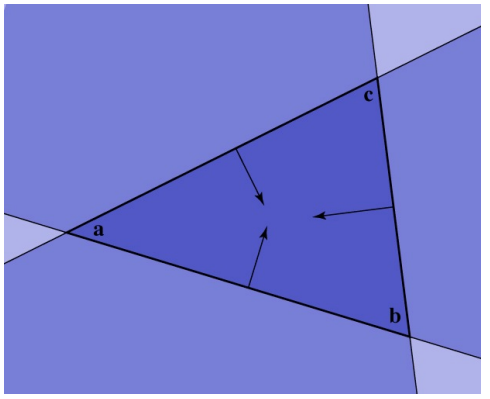
15

Ray-triangle intersection

$$(\mathbf{b} - \mathbf{a}) \times (\mathbf{x} - \mathbf{a}) \cdot \mathbf{n} > 0$$

$$(\mathbf{c} - \mathbf{b}) \times (\mathbf{x} - \mathbf{b}) \cdot \mathbf{n} > 0$$

$$(\mathbf{a} - \mathbf{c}) \times (\mathbf{x} - \mathbf{c}) \cdot \mathbf{n} > 0$$



or, more generally, they must all have the same sign
(if normal can be in either direction)

16

Ray-triangle intersection

- See book for a more efficient method based on linear systems

other objects: triangle mesh

to do more complex objects, you'll need a structure for a collection of triangles: a **triangle mesh**

this will allow you to speed up intersections by grouping adjacent geometry

here is the naive implementation:

```
class tri_mesh  {
        std:vector<tri> triangles_;
        int nt_;
};
```

smarter implementations allow adjacent triangles to:

- share vertices (and thereby reduce storage)
- share normals (which sounds counterintuitive, but allows smooth shading of triangular geometry)

other simple geometric objects

parallelpipeds - (collection of planes)

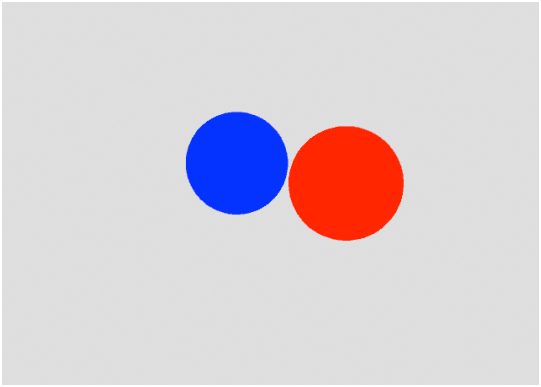
cones, cylinders, disks, paraboloids - (quadrics, similar to sphere)

convex polygon - (similar to triangle)

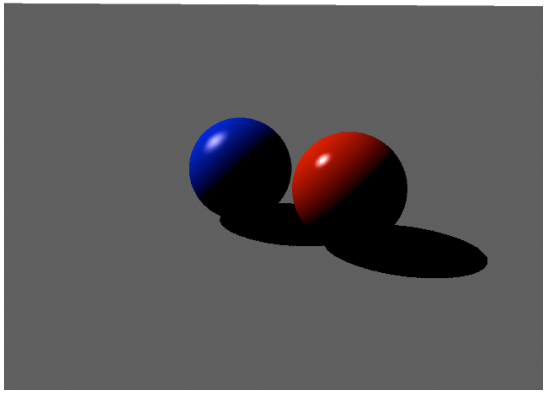
concave polygon - (similar to plane but has more complex inclusion test)

with boolean operations and transformations, these classes will provide even more shape flexibility

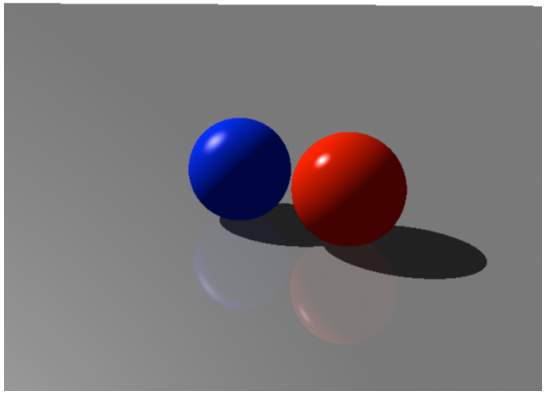
simplest "shading" - fixed color only



phong shading (diffuse + specular) + shadows



phong shading (diffuse + specular) model + shadows + ambient + true specular



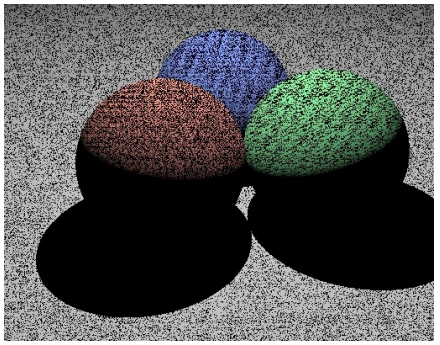
shadows

a surface is only illuminated if nothing blocks its view of the light.

With ray tracing it's easy to check

- just intersect a ray with the scene!
- this ray starts at the hit point, and points towards the light source
- if it does not intersect any surfaces in the scene, there must be an unoccluded path from the light to the hit point, and so the energy from that light can be used in the shading calculation

shadow rounding errors



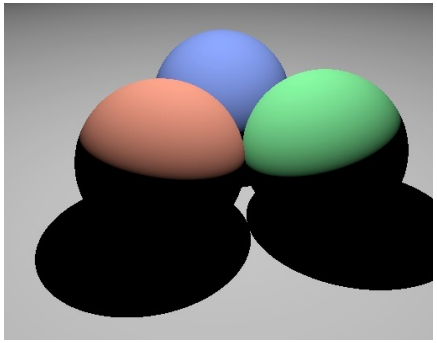
What's going on?

- hint: at what t does the shadow ray intersect the surface you're shading?

slide courtesy steve marschner (author of our textbook)

shadow rounding errors

Solution: shadow rays start a tiny distance from the surface



Do this by moving the start point, or by limiting the *t* range

slide courtesy steve marschner (author of our textbook)

multiple lights

important to fill in black shadows

just loop over lights, add contributions

(can add ambient shading if no lights contribute)

- black shadows are not really right
- one solution: dim light at camera
- alternative: add a constant “ambient” color to the shading, based on the diffuse reflectance of the object and the spd of the light…

slide courtesy steve marschner (author of our textbook)

types of lights

point

directional

area

spot (w “flaps”)

volumetric

“ambient”

simple point light class

```
class p_light {

    point position;

    colorRGB color;  // we'll use [0..1] for each
    channel

    float intensity;  // simple multiplier on color

};


// note, not a "surface" or in surface list, so it
// doesn't get hit by rays
```

simple directional light class

```
class d_light {
      vector dir;
      colorRGB color;  // we'll use [0..1] for each channel
      float intensity;  // optional simple multiplier on color
};

// note, not a "surface" or in surface list, so it
// doesn't get hit by rays
// use cos(dir . dir_to_hit_point) to scale intensity
```


Putting it together

Usually include ambient, diffuse, Phong in one model

$$L = L_a + L_d + L_s$$
$$= k_a\, I_a + k_d\, I \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s\, I \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

The final result is the sum over many lights

$$L = L_a + \sum_{i=1}^{N} [(L_d)_i + (L_s)_i]$$

$$L = k_a\, I_a + \sum_{i=1}^{N} [k_d\, I_i \max(0, \mathbf{n} \cdot \mathbf{l}_i) + k_s\, I_i \max(0, \mathbf{n} \cdot \mathbf{h}_i)^p]$$

slide courtesy steve marschner (author of our textbook)

assignment: theme 1, milestone 3

build up on your past assignment:

- triangles!
- shadows!
- multiple lights!

scene file description

The command file consists of a series of lines which describe geometry, camera, lights, or materials.

All points, scalars, or vectors are given as floats, with distances in mm. r/g/b values are encoded as floats with range [0 1] for material colors, and light color and intensity are both encoded in and [r g b] triple, with minimum 0 and unbounded maximum (although it's reasonable to choose 1 as a nominal value).

Comment:

/ Any line starting with / should be ignored

Geometry:

/ sphere at position x y z with radius r:
s x y z r

/ triangle with counterclockwise point order:
t x1 y1 z1 x2 y2 z2 x3 y3 z3

/ plane with normal n and scalar value d:
p nx ny nz d

"Knickknack"

- pixar animation studios, 1989
- rendered using PR Renderman:
  - pipeline renderer, models reflectance as diffuse transport
  - primitives: polygons, degree-1,2&3 curves & surfaces
  - uses "REYES" architecture: all scene objects are subdivided to polygons smaller than a pixel
  - hard shadows, simple shapes textures & colors, Gouraud shading (imitates specular component in a pipeline renderer of this vintage)