CS4160

COMPUTER GRAPHICS

class 3

today's class

ray tracing - basic structures

intro to materials

some notes from signal processing

hw 1.1 overview

animation appreciation:
"The Adventures of Andre and Wally B."

Ray Tracing

introduction

basic idea:

- trace the path light follows though a scene.
- model its material interactions.
- iterate. a lot.

introduction

strength of the algorithm:

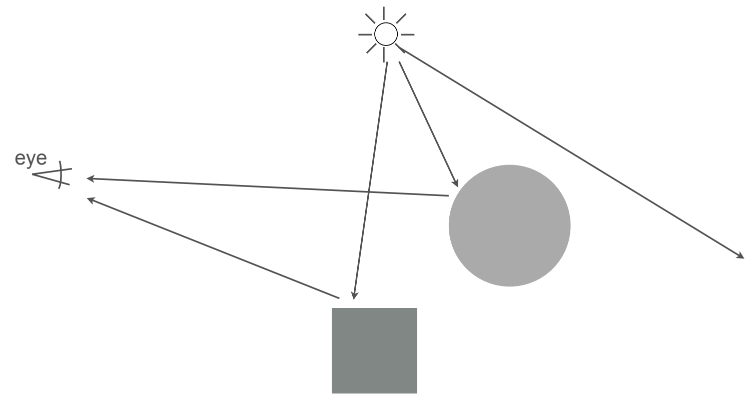
- very intuitive solution to visible surface determination
- extends to many secondary phenomena (i.e. shadows, refraction, multiple lights, etc.)
- produces very realistic images

weaknesses:

- not good for some common secondary phenomena (diffuse inter-reflection)
- slow

5

geometric optics - light propagating through a scene

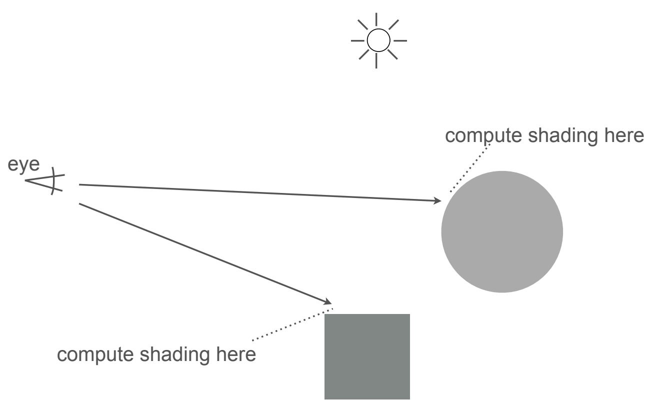


big idea: light "rays" follow simple geometric rules when they reflect around a scene

problem: almost all the rays dont end up at the eye (or film)

6

ray tracing - big idea



7

simple ray tracing - pseudocode

for each pixel:

compute the ray through the current pixel

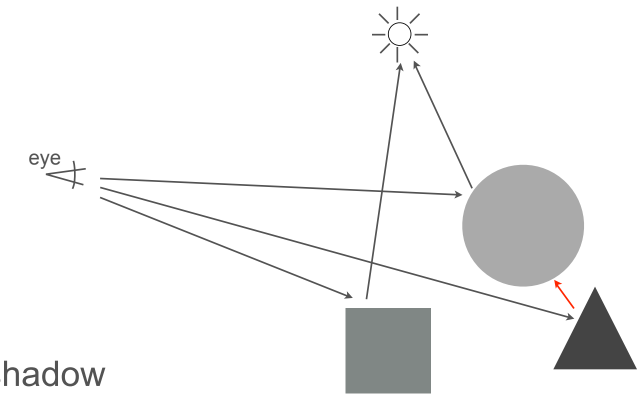
intersect ray with the scene

compute shading at intersection point

put result into current pixel

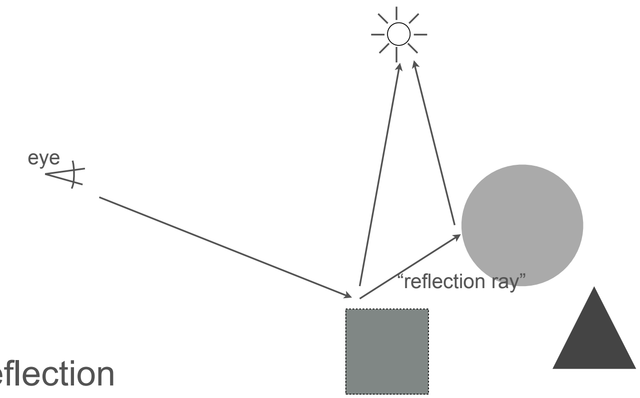
8

slightly more advanced ray tracing



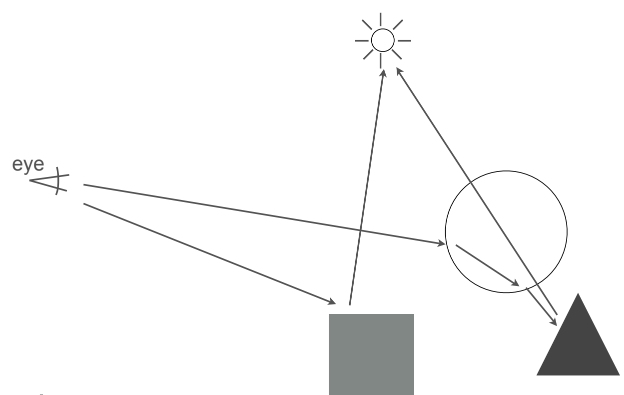
shadow
rays

slightly more advanced ray tracing



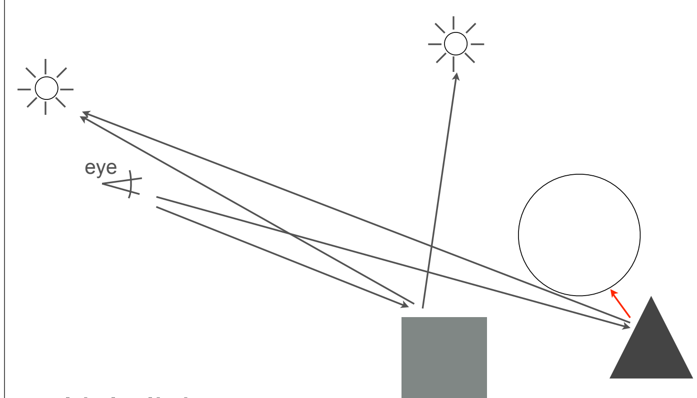
reflection
rays

slightly more advanced ray tracing



refraction rays

slightly more advanced ray tracing



multiple lights

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

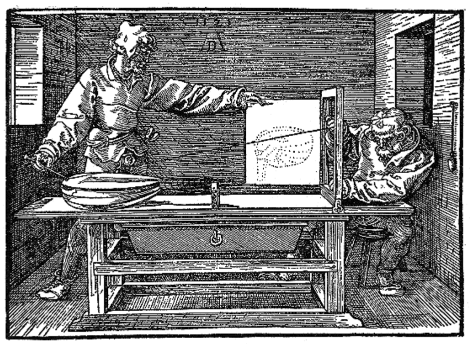
(if ray missed all objects, set color to background value)

benefits of ray tracing

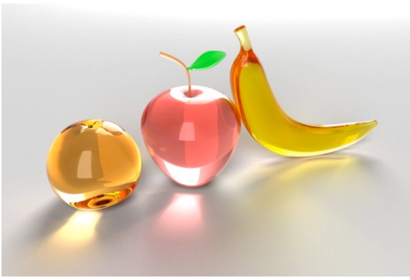
aside from it's natural extensibility to handle shadows, reflection, refraction, etc:

- raytracing provides an easily-understood solution to visibility determination from any point in the scene
- automatically generates images in perspective (because the primary rays may be constructed to go through a single focal point)

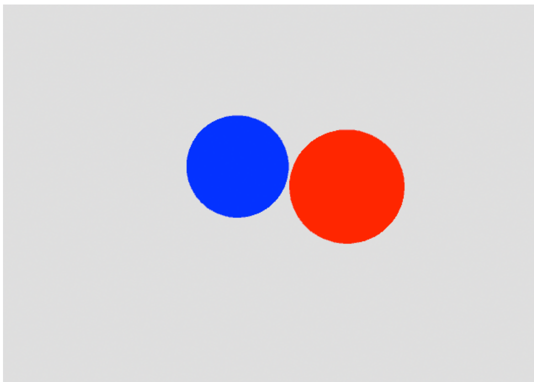
perspective



ray traced example scene

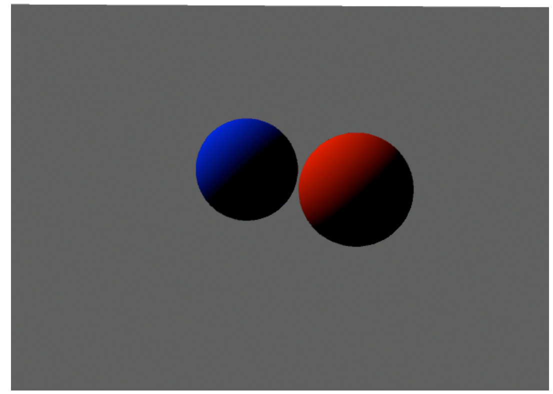


simplest "shading" - fixed color only



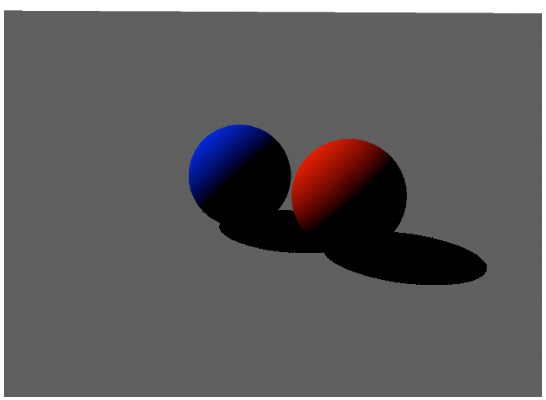
17

diffuse (lambertian) shading



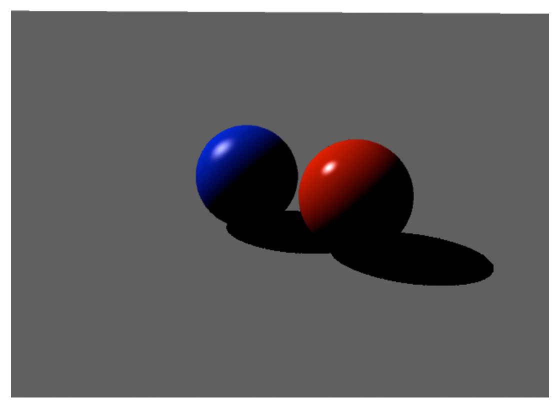
18

diffuse + shadows



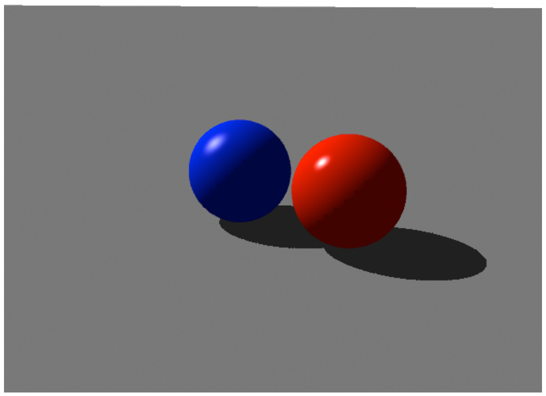
19

phong shading (diffuse + specular) + shadows



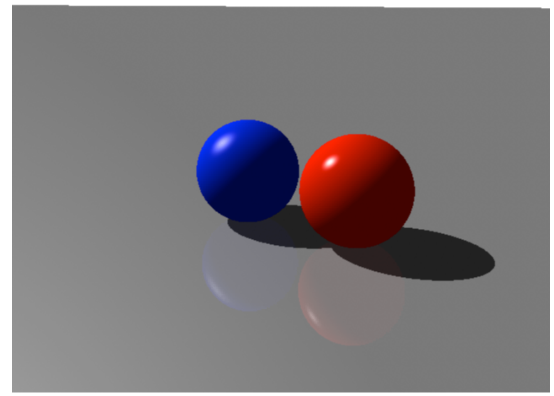
20

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



21

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



22

modeling the system

what we'll need:

- a way to represent and work with "rays"
- a camera model
- geometric models - for the "objects" in our synthetic scene
- a way to represent lights
- a shading model - simple material descriptions
- + support classes: image, point/vector/intersection/etc which have the obvious methods (normalize, point-subtract, etc.)

23

rays

what they are:

- geometric interpretations of the path light follows while propagating through a scene

what you want to do with them:

- generate them according to camera model or other data
- test them for intersection with objects in the scene (and determine which object is closest)

24

rays

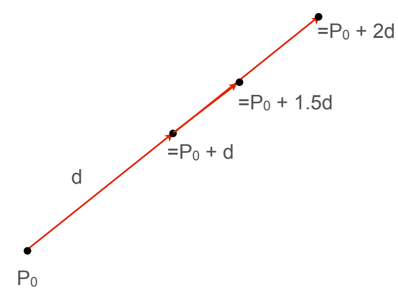
rays are defined via linear interpolation in the form:

$$\mathbf{r}(t) = \mathbf{P_0} + t\mathbf{d}$$

where "**d**" is a vector and $\mathbf{P_0}$ is a point.

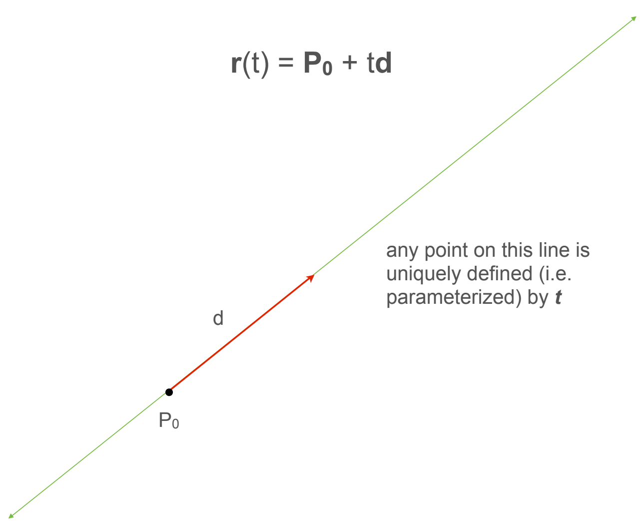
rays

$$\mathbf{r}(t) = \mathbf{P_0} + t\mathbf{d}$$



rays

$$\mathbf{r}(t) = \mathbf{P_0} + t\mathbf{d}$$



simple ray class

```
class ray {

   point origin;
   vector dir;

};
```

constructing rays

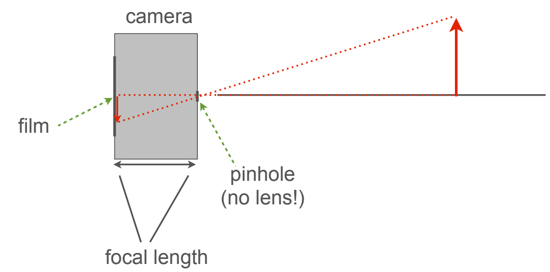
from two points $\mathbf{p_1}$ and $\mathbf{p_2}$:

- ray origin = $\mathbf{p_1}$, ray dir = $\mathbf{p_2} - \mathbf{p_1}$
- **OR:** ray origin = $\mathbf{p_2}$, ray dir = $\mathbf{p_1} - \mathbf{p_2}$
- **normalize dir!**

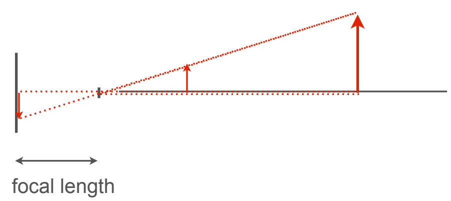
from point $\mathbf{p}$ and vector $\mathbf{v}$:

- ray origin = $\mathbf{p}$, ray dir = $\mathbf{v} / ||\mathbf{v}||$

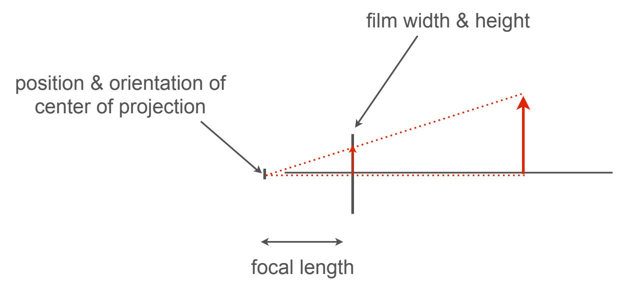
pinhole camera model



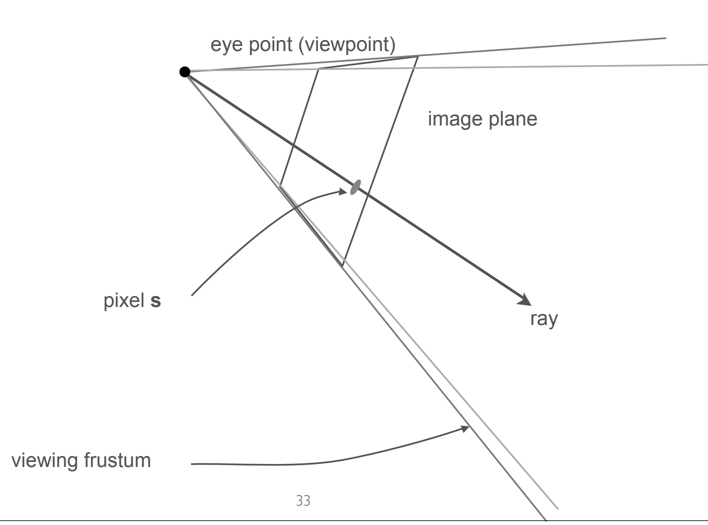
abstract camera model



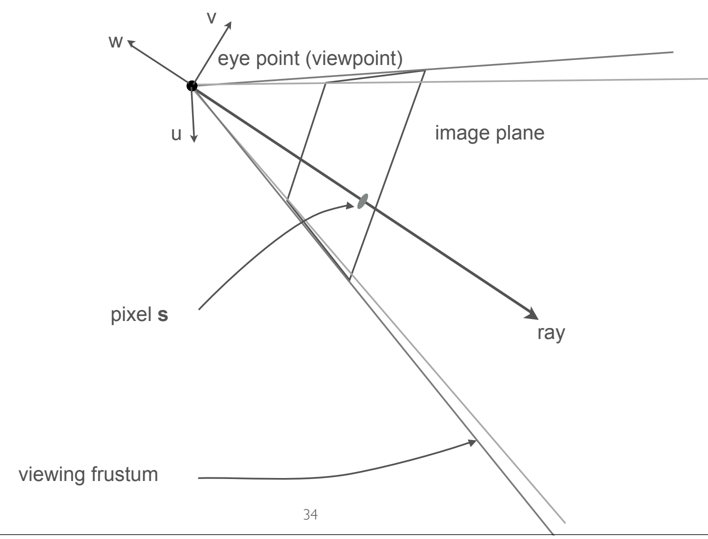
abstract camera model



camera model



camera coordinate frame



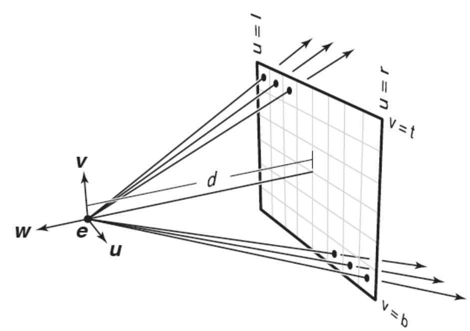
camera coordinate frame

the camera coordinate frame consists of:

the eye point
vector w (opposite the viewing direction)
vector u (the camera's "right")
vector v (the camera's "up")

note: this coordinate frame is orthonormal
(u, v, w are normalized, and u x v = w, w x u = v, v x w = u)

camera coordinates



e is the eye point, with vectors **uvw** form the "camera coordinate frame"
d is the focal length (note: in direction -***w***)

generating rays for a perspective camera

to construct the ray through pixel (i,j):

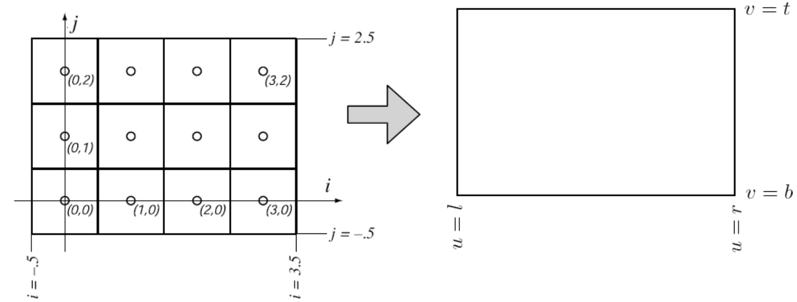
compute *u* and *v* for the pixel center

ray dir = -d**w** + *u***u** + *v***v**   (normalize this!)

ray origin = **e**

(note: d is the focal length)

relationship of *(i, j)*th pixel in image plane to *u*,*v*

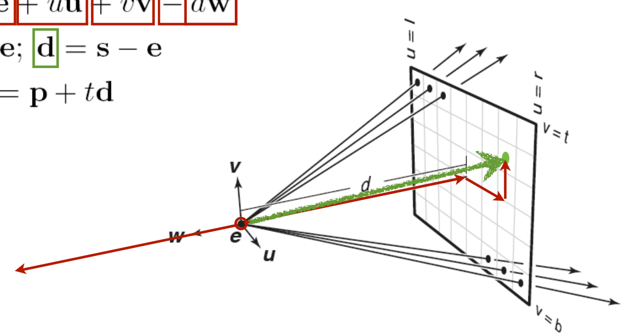


$$u = l + (r - l)(i + 0.5)/n_x$$
$$v = b + (t - b)(j + 0.5)/n_y$$

note: n_x, n_y are the width, height of the image in pixels. *l, r, t, b*, are the left,right,top, bottom edges of the screen in *u,v* coordinates (usually, an image is centered and *l*=-*r* and *b* = -*t*). All are scalars.

camera coordinates

$$\boxed{\mathbf{s}} = \boxed{\mathbf{e}}\boxed{+\,u\mathbf{u}}\boxed{+\,v\mathbf{v}}\boxed{-\,d\mathbf{w}}$$
$$\mathbf{p} = \mathbf{e};\ \boxed{\mathbf{d}} = \mathbf{s} - \mathbf{e}$$
$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$



d is the focal length, **d** is the desired ray direction
e is the eye point, **s** is the “pixel-of-interest’s” center
e, with vectors **uvw** form the “camera coordinate frame”

simple camera class

```
class camera {

    point eye;

    float d;     // focal length

    vector u;
    vector v;
    vector w;

    int nx;
    int ny;

    float l, r, t, b; // or just w,h
                      // if film is centered

};
```

[as well as methods to construct a ray given a pixel (i,j)]

constructing a camera frame

often times you are given

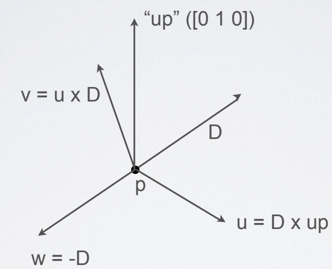
- an eye point **p**, and
- a direction "**D**"

and need to create a camera frame.

use

- D x "UP" [0 1 0] to form u
- then u x D to form v
- w = -D
- normalize everything
- **n.b.** - this won't work when D is [0 s 0], where s is arbitrary

constructing a camera frame



n.b. all vectors must be normalized

modeling the scene

to model objects in the scene, we need:

- precise descriptions of their boundaries
- methods for computing their intersection with a ray (as fast as possible)
- a way to represent their material properties (see later section on materials)

we also need to be able to keep lists, trees, queues, and other data structures of these objects

so it's a good idea to make these subclasses of a general "`surface`" class, which we will extend later

planes

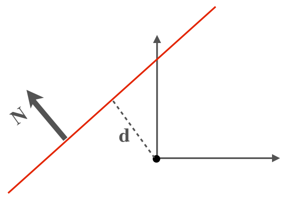
planes can be used to represent limitless ground (i.e. surface of the "world")

they can be useful as a bounding surface in acceleration structures

planes

implicit equation for a plane with normal **N** and distance to origin **d:** (n.b. **d** is measured in direction of normal!)

$$\mathbf{p}\cdot\mathbf{N}+\mathbf{d} = 0$$



45

planes

```
class plane : public surface {
  vector normal;
  float d;   // distance to origin
};
```

46

ray-plane intersection

intersection of plane with a ray r(t):

$\mathbf{p}\cdot\mathbf{N}+\mathbf{d} = 0$

$\mathbf{r}(t) = \mathbf{P_0} + t\vec{\mathbf{d}}$ (note: renaming ray vector to d to avoid conflict with plane scalar d)

Solution:

$t = -(\mathbf{P_0} \cdot \mathbf{N} + \mathbf{d}) / \vec{\mathbf{d}}\cdot\mathbf{N}$ (solve for t)

$\mathbf{p} = \mathbf{P_0} + t\vec{\mathbf{d}}$ (plug t back into ray equation to get **p**)

note: if $\vec{d}\cdot N = 0$, ray and plane are parallel

normal at intersection point is **N** (n.b. we'll need the normal for shading calculation)

47

planes

creating a plane from 3 points p_1 p_2 p_3:

1. use cross products of (p_2-p_1) and (p_3-p_1) to compute normal $N = [N_1\ N_2\ N_3]$

2. put any of p_1 p_2 p_3 into $\mathbf{p}\cdot\mathbf{N} = -\mathbf{d}$

48

spheres

spheres are useful as:

- test objects
- bounds for other (more complex) objects
- representations of sky

49

49

simple sphere class

```
class sphere : public surface {
  point O;   // origin
  float r;    // radius
};
```

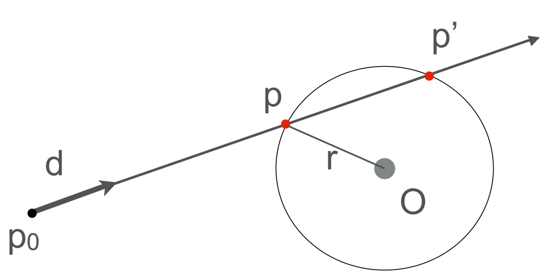
50

50

spheres - intersections

ray: **r**(t) = $\mathbf{P_0}$ + t**d**

sphere: (p-O)·(p-O) - r^2 = 0

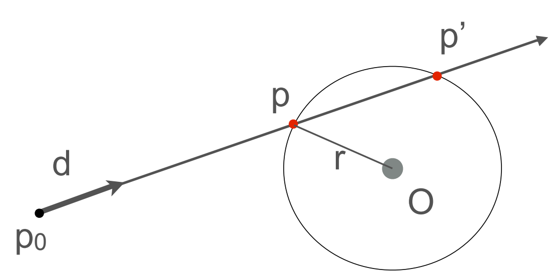


51

51

spheres - intersections

$$t = \frac{-d \cdot (p_0 - O) \pm \sqrt{(d \cdot (p_0 - O))^2 - (d \cdot d)((p_0 - O) \cdot (p_0 - O) - r^2)}}{d \cdot d}$$



52

52

spheres - intersections

$$t = \frac{-d \cdot (p_0 - O) \pm \sqrt{(d \cdot (p_0 - O))^2 - (d \cdot d)((p_0 - O) \cdot (p_0 - O) - r^2)}}{d \cdot d}$$

note:

- descriminant negative: line & sphere do not intersect
- descriminant zero: 1 intersection (graze)
- otherwise, two intersects

unit normal at intersection is (p-O)/r

intersections

for ray/surface intersection tests, we need to record:

- yes/no intersection
- "t" - the parameterization of the intersection point
- "t_2" (a second intersection for quadratics) to save time
- **intersection point** - for shading
- **geometric normal** - also for shading
- surface id - so we can look up materials
- all this suggests an intersection class

take-aways from last class:

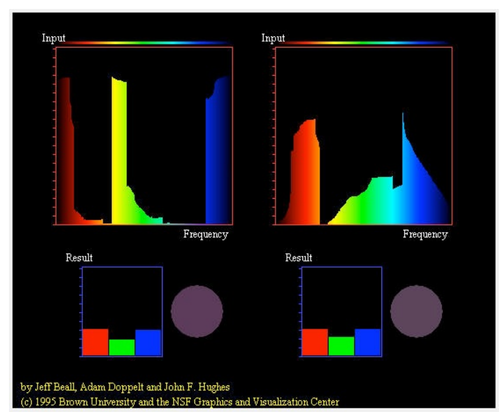
in this class, we will almost always be discussing light in terms of geometric optics, which models light as a particle (i.e. photons)

due to the existence of metamerism, we can use RGB values to represent colors

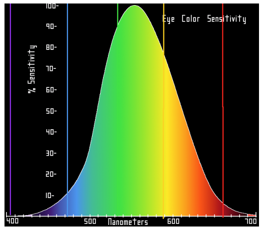
we can represent each channel ({R, G, B}) as a single type: say "unsigned 8 bit integer" or "unsigned 16-bit float"

metamers

metamers are two emitters (or reflectors) with different SPDs, but which appear the same (i.e. perceptually).



hw0 note: luminance - RGB to greyscale



the human eye is not equally sensitive to R, G, & B!

this does not properly compute the relative grayscale value:

~~L = .33R + .33G + .33B~~

a reasonable (empirical) formula is:

L = .21R + .71G + .07B

materials & visual phenomena

visual phenomena

light is emitted from sources

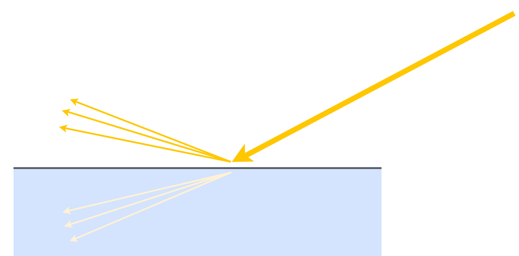
light interacts with (objects in) the scene

- some is absorbed
- some is scattered in new directions

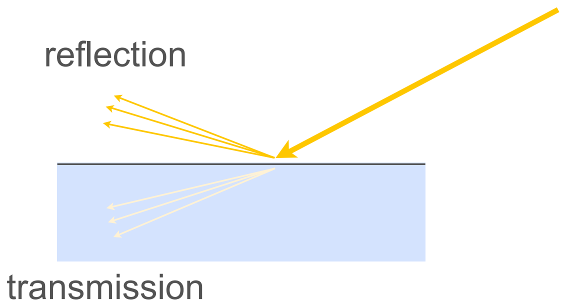
some light enters sensor and is recorded

- eye, CCD, film, lightmeter, etc.

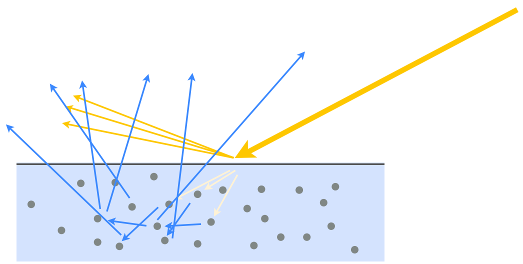
scattering & absorption



scattering & absorption



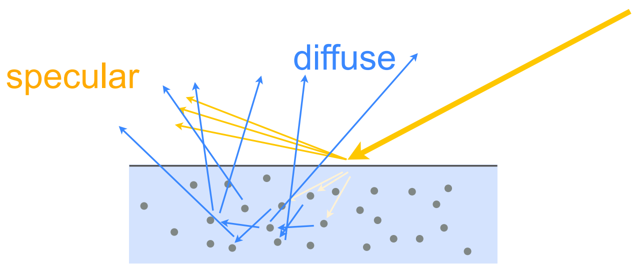
scattering & absorption in opaque objects



some light is reflected, but the rest is absorbed, has multiple scatter events, and is re-emitted

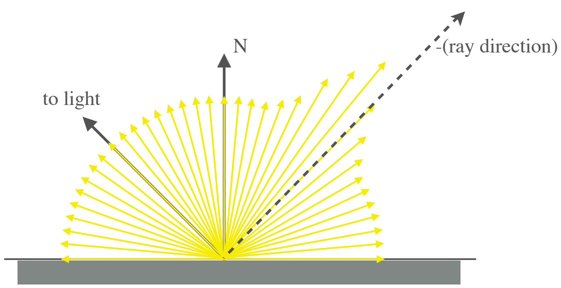
(note new directions and color of the emitted light)

scattering & absorption



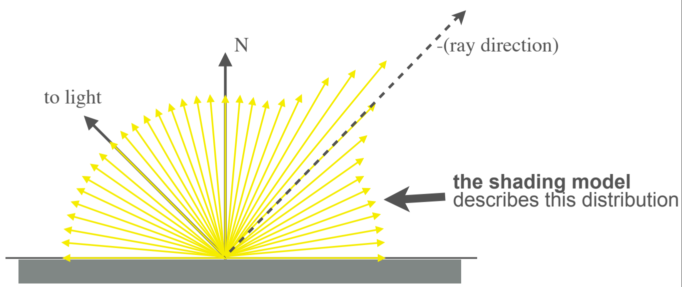
how do we characterize reflectance geometrically?

the angle of the ray to the surface
the angle of the light to the surface



how do we characterize reflectance geometrically?

the angle of the ray to the surface
the angle of the light to the surface
how these affect the exiting energy are descirbed by the **shading model**



material description

at (just about) its simplest, the material description defines:

specular reflection as an RBG triple *1

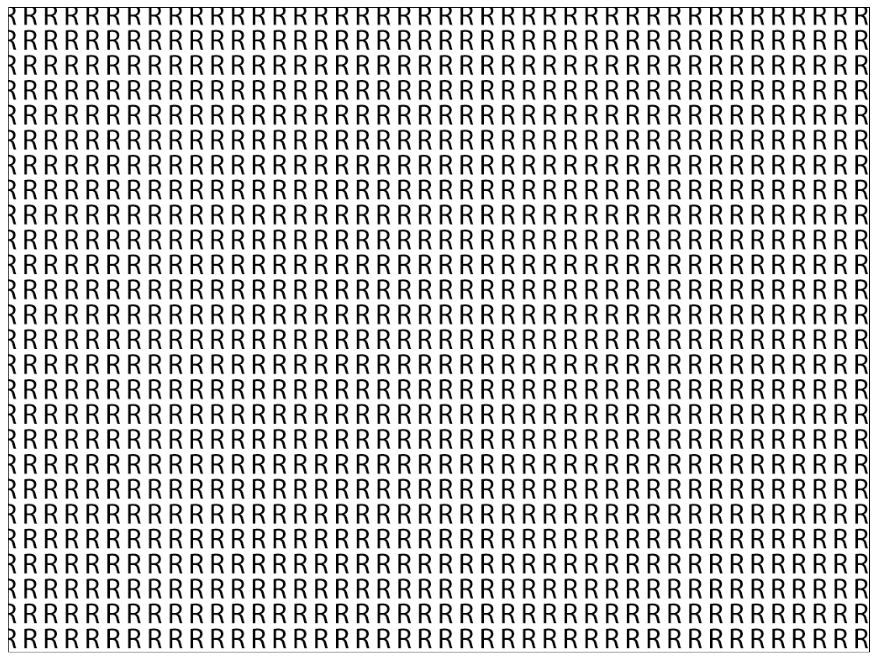
diffuse reflection as an RBG triple *1

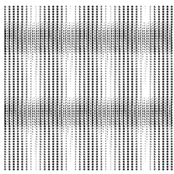
the shading combines these with an RGB triple that describes the incident light energy *2 to produce an RGB triple describing the emitted energy along a ray *2

*1: these on [0 1]

*2: these may be > 1

images & signal processing

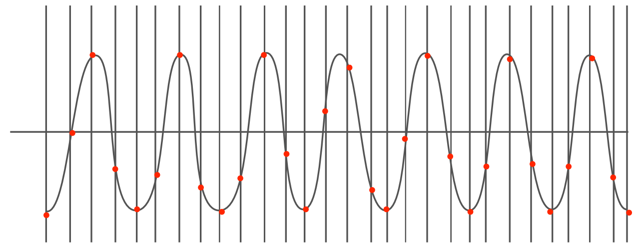




images are digital signals, and as such have the same behaviors, and are bound by the same limitations, as other signals

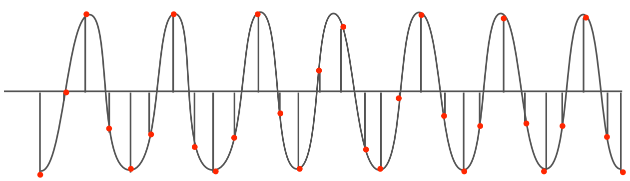
69

sampling a 1d signal



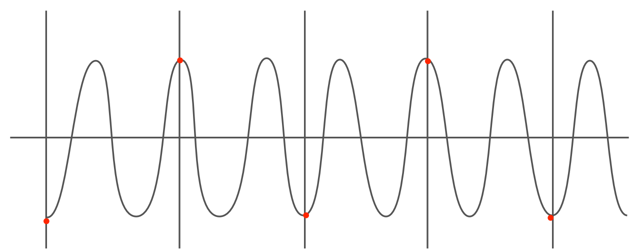
70

sampling a 1d signal



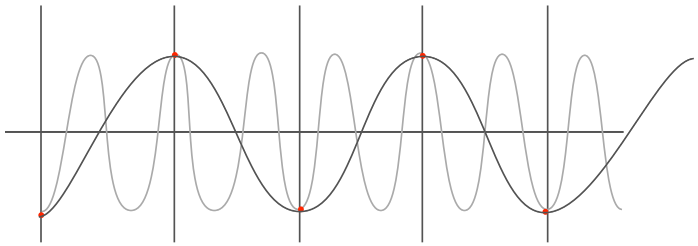
71

sampling a 1d signal



72

sampling a 1d signal

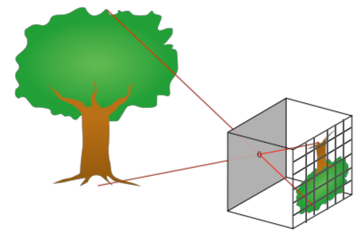


73

image sampling

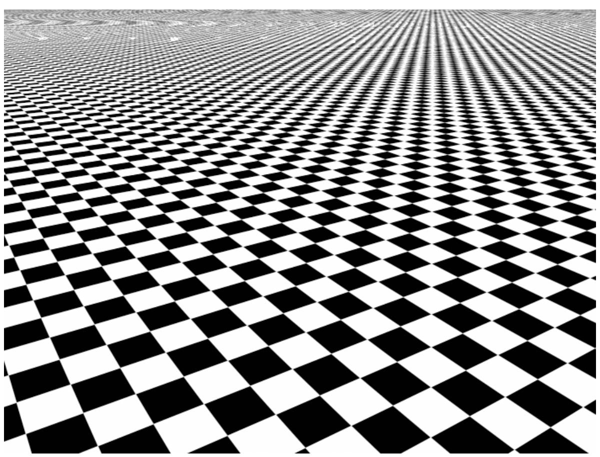
create image from a continuous function

- i.e. create a discrete representation from a continuous one
- 2-D example: image formation on camera sensor



74

example: discrete image from continuous function (note aliasing)



75

assignment 1.1

write, test, and evaluate a raytracing renderer named **raytra**

eventually, this will be a well-featured renderer, including:

- multiple light sources & types
- multiple geometry types
- diffuse, specular, & mirror reflection models
- refraction on transparent materials
- spatial acceleration structures

76

assignment: theme 1, part 1

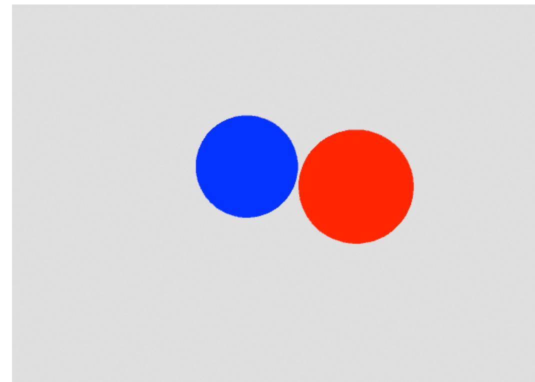
for this first part, implement only the most basic functionality:

- scene file read (we provide a parser for you!)
- camera & image setup
- primary ray generation
- ray/sphere intersection (note: there may be multiple spheres)
- **no**: shading, materials, lights, etc.

for any pixel, if there is a ray-sphere intersection, the color for that pixel should be red

if there is no ray ray-sphere intersection, the color of that pixel should be black

simplest “shading” - fixed color only



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

scene file description

Camera:

/ camera at position [x y z] looking in direction [vx vy vz], with focal length d,
/ an image plane sized iw by ih (width, height) and number of pixels pw ph.
c x y z vx vy vz d iw ih pw ph

Lights: (note second parameter to denote which kind of light)

/ a point light at position x y z, color & intensity coded as r g b
l p x y z r g b

/ a directional light with direction vx vy vz and color & intensity coded as r g b
l d vx vy vz r g b

/ the ambient light (there will be, at most, only one of these):
l a r g b

scene file description

Materials:

/ set the geometry's material to be this one, (applies to geometry defined
/ after this statement)
/ defined by diffuse components [dr dg db] and specular components
/ [sr sg sb], ideal specular components [ir ig ib], and with "roughness"
/ or phong exponent "r"

m dr dg db sr sg sb r ir ig ib

animation appreciation:

"The Adventures of Andre and Wally B."
Pixar, 1984