CS4160

COMPUTER GRAPHICS

class 7

today's class

creative submissions!

profiling code

reflections and refractions

code walkthrough for 1.2

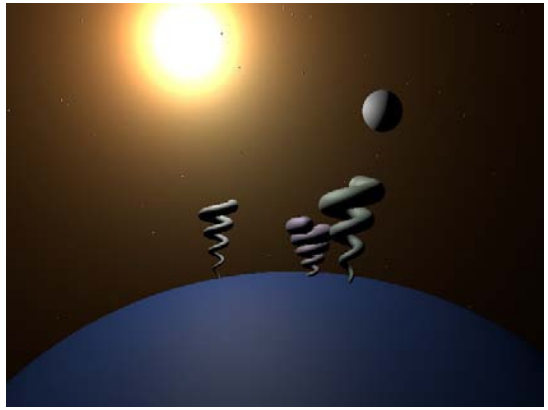
creative image submission: raytra 1.1

creative image submission: raytra 1.1

iris zhang - 3rd place

creative image submission: raytra 1.1

narcisa codreanu - 2nd place



creative image submission: raytra 1.1

mehmet turcan - 1st place!

profiling code

graphics is a very "optimization-centric" culture

the fast algorithm, and the faster implementation, make better:

- images
- products
- papers

in rendering, so many operations are repeated, sloppiness will be really evident!

because of this, profiling is very common - use it to identify where time is being taken up.

profiling code

via:

- external tools:
  - instrumented during compilation
  - or sampled during execution
- or: permanently instrument your code

- three easy ways:
  - UNIX "time" command
  - gprof statistical profiler (-pg option on g++)
  - getrusage (instrument your code yourself)

gprof

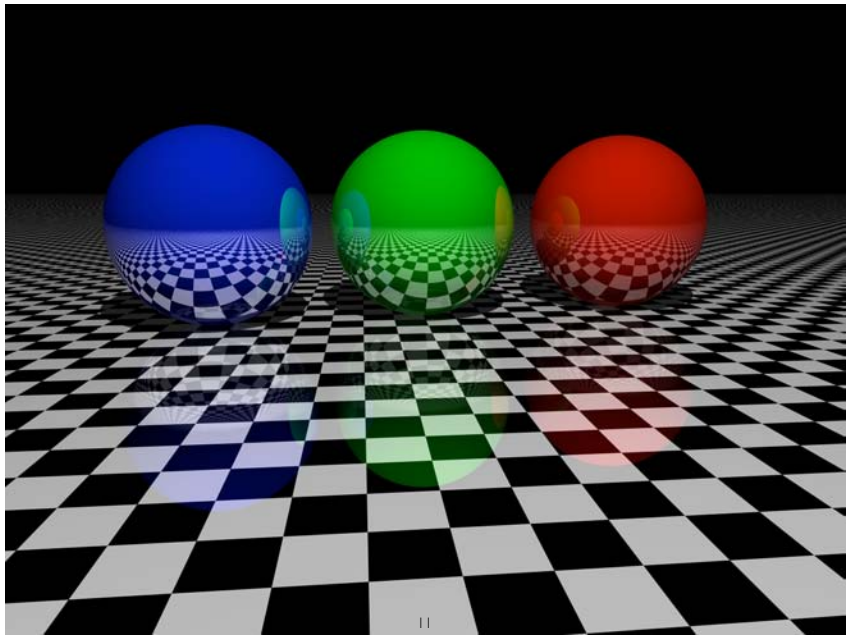
gprof (profiler)

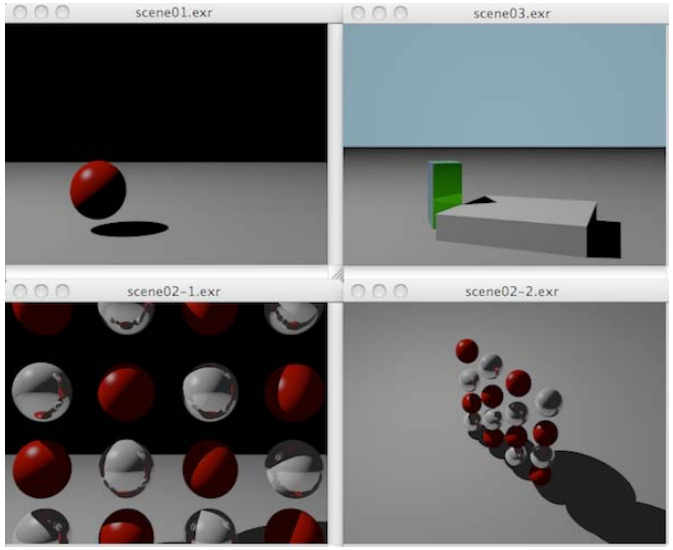
- compile with -pg
- run exectutable
- process output file with gprof

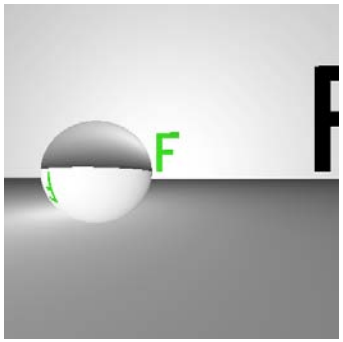
gprof output

```
Flat profile:

  Each sample counts as 0.01 seconds.
    %   cumulative   self              self     total
   time   seconds   seconds    calls  ms/call  ms/call  name
   33.34      0.02     0.02     7208     0.00     0.00  open
   16.67      0.03     0.01      244     0.04     0.12  offtime
   16.67      0.04     0.01        8     1.25     1.25  memccpy
   16.67      0.05     0.01        7     1.43     1.43  write
   16.67      0.06     0.01                             mcount
    0.00      0.06     0.00      236     0.00     0.00  tzset
    0.00      0.06     0.00      192     0.00     0.00  tolower
    0.00      0.06     0.00       47     0.00     0.00  strlen
    0.00      0.06     0.00       45     0.00     0.00  strchr
    0.00      0.06     0.00        1     0.00    50.00  main
    0.00      0.06     0.00        1     0.00     0.00  memcpy
    0.00      0.06     0.00        1     0.00    10.11  print
    0.00      0.06     0.00        1     0.00     0.00  profil
    0.00      0.06     0.00        1     0.00    50.00  report
  ...
```







Mirror reflection

Consider perfectly shiny surface

- there (usually) isn't a highlight
- instead there's the reflection of other objects

Can render this using recursive ray tracing

- to find out mirror reflection radiance ("color,") ask what color is seen from surface point in reflection direction

shiny material has mirror reflection and diffuse

$$L = L_a + L_s + L_d + L_m$$

- where L_m is evaluated by tracing a new ray

mirror reflection

things to note:

You must generate a new ray that returns an spd ("color") value from its scene intersection & lighting

- (that value is actually termed "radiance" - the amount of energy that travels along a ray and is constant over distance)

you must limit the number of these recursive raytracing calls

- one easy way: limit the depth of recursion.

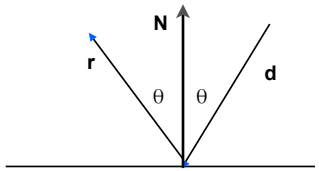
$\mathbf{L_m = k_m}$ * energy_along_ray (r, p)

(where r is the computed, reflected ray, and p is the intersection point, and "*" is component-wise multiply)

Mirror reflection

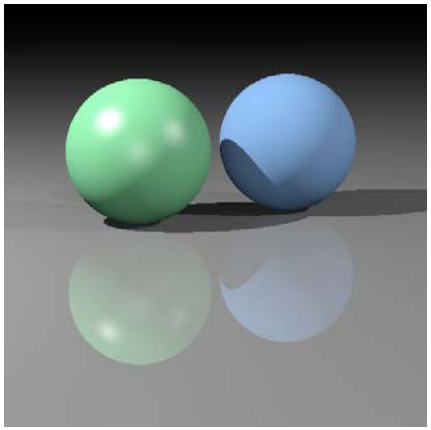
like phong specular - depends on view direction

- reflects incident light from mirror direction



$$\mathbf{r} = \mathbf{d} - 2(\mathbf{d}\cdot\mathbf{N})\mathbf{N}$$

diffuse + mirror reflection



(material on floor)

recursive ray tracing and true specular reflection

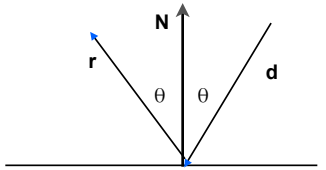
add one more coefficient for reflective materials: $\mathbf{k}_m$

compute reflected ray **r**

$\boldsymbol{L}$ = [diffuse + specular + ambient terms] + $\mathbf{k}_m$ * ray_code (reflected ray)

(note: if $\mathbf{k}_m$ = [0 0 0], there is no reflection)

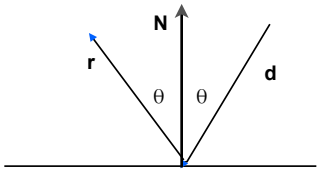
(note: limit # of recursions!)



to compute reflected ray r

(slightly different formulation than 3 pages prior)

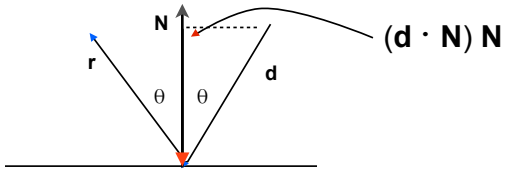
$\mathbf{r} = \mathbf{d} - 2\,(\mathbf{d} \cdot \mathbf{N})\,\mathbf{N}$



to compute reflected ray r

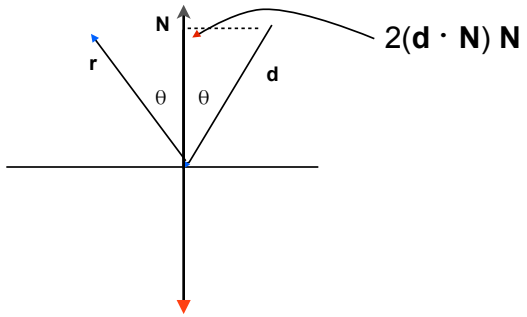
(slightly different formulation than 3 pages prior)

$\mathbf{r} = \mathbf{d} - 2\,(\mathbf{d} \cdot \mathbf{N})\,\mathbf{N}$



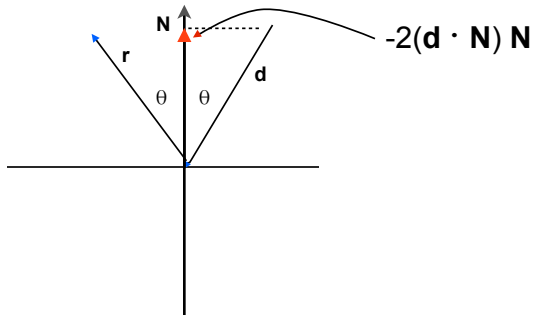
to compute reflected ray r

$\mathbf{r} = \mathbf{d} - 2\,(\mathbf{d} \cdot \mathbf{N})\,\mathbf{N}$



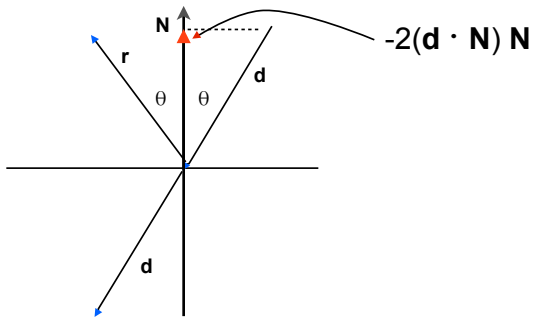
to compute reflected ray r

$\mathbf{r} = \mathbf{d} - 2\,(\mathbf{d} \cdot \mathbf{N})\,\mathbf{N}$



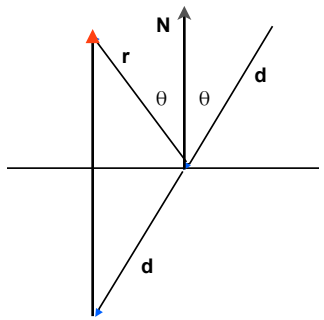
to compute reflected ray r

$\mathbf{r} = \mathbf{d} - 2\,(\mathbf{d} \cdot \mathbf{N})\,\mathbf{N}$



to compute reflected ray r

$\mathbf{r} = \mathbf{d} - 2\,(\mathbf{d} \cdot \mathbf{N})\,\mathbf{N}$



raytracing refractions

transparency (for dielectrics)

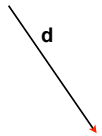
transparency (for dielectrics)

index of refraction for common materials

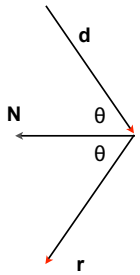
vacuum	1.0	
air	1.000029	(at sea level)
water	1.33	
glycerin	1.47	
glass	1.50	
quartz crystal	1.54	
diamond	2.42	

refraction - snell's law

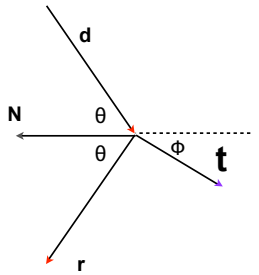
refraction - snell's law



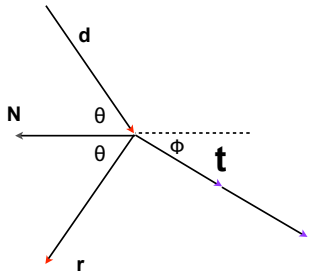
refraction - snell's law



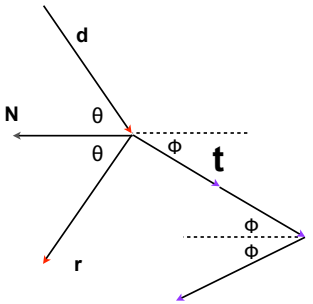
refraction - snell's law



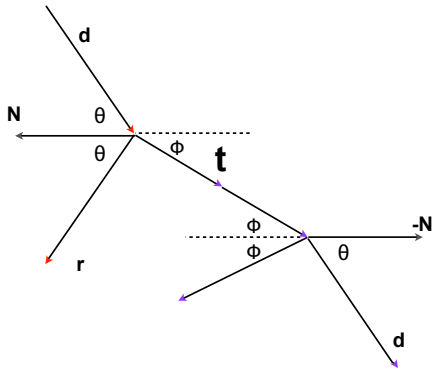
refraction - snell's law



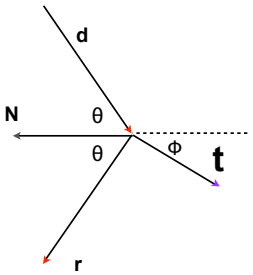
refraction - snell's law



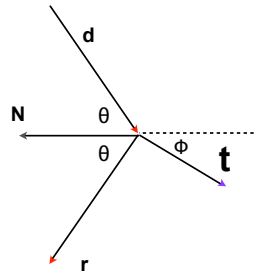
refraction - snell's law



refraction - snell's law



refraction - snell's law

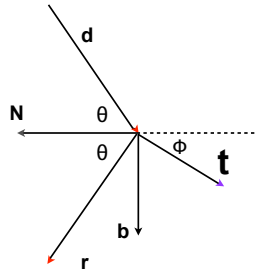


$n \sin \theta = n_t \sin \Phi$ (n, n_t are the indices of refraction, with $n_t > n$)

refraction - developing transmitted ray in 3d

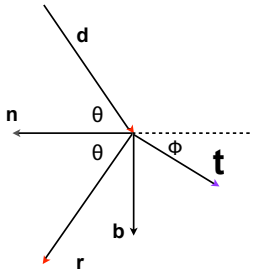
t = *sin*Φ **b** - *cos*Φ **N**

d = *sin* θ **b** - *cos* θ **N**



refraction - developing transmitted ray in 3d

$$t = \frac{n(d - N(d \cdot N))}{n_t} - N\sqrt{1 - \frac{n^2(1-(d \cdot N)^2}{n_t^2}}$$



note: if term inside square root is < 0, total reflection (and no refraction)

convert snell's law to cosines: (as they can be expressed as dot products)

$$nsin\theta = n_t sin\phi$$

$$n^2 sin^2\theta = n_t^2 sin^2\phi$$

$$n^2(1 - cos^2\theta) = n_t^2(1 - cos^2\phi)$$

$$cos^2\phi = 1 - \frac{n^2(1 - cos^2\theta)}{n_t^2}$$

$$cos\phi = \sqrt{1 - \frac{n^2(1 - cos^2\theta)}{n_t^2}}$$

$$d = \sin\theta b - \cos\theta N \qquad \therefore \quad b = \frac{d + \cos\theta N}{\sin\theta}$$

$$T = \sin\phi b - \cos\phi N$$

$$T = \frac{\sin\phi(d + \cos\theta N)}{\sin\theta} - N(\sqrt{1 - \frac{n^2(1 - \cos^2\theta)}{n_t^2}})$$

$$T = \frac{n\sin\theta(d + \cos\theta N)}{n_t\sin\theta} - N(\sqrt{1 - \frac{n^2(1 - \cos^2\theta)}{n_t^2}})$$

$$T = \frac{n(d - N(d \cdot N))}{n_t} - N(\sqrt{1 - \frac{n^2(1 - (d \cdot N)^2)}{n_t^2}}) \quad \blacksquare$$

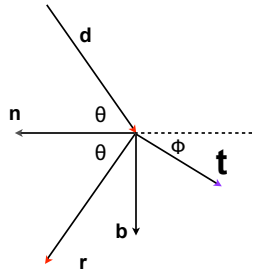
how much light is reflected or refracted?

reflectivity is determined by the Fresnel equations, and varies with the incident angle, material type, and polarization!

a reasonable solution is given by the Schlick approximation, describing the reflected %:

$R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5$

with $R_0 = ((n_t-1)/(n_t+1))^2$



(1 - R(θ)) is the refracted %.

how does the light's color change during refraction?

as the light transmits through the medium, the intensity is attenuated according to Beer's law:

$$I(s) = I_0 e^{\ln(a)s}$$

where: $\mathbf{I_0}$ is the radiance (an rgb triple - often on [0 1] but can be greater) at the boundary, **s** is the distance travelled through the medium, and a is an attenuation constant (an rgb triple, definitely on (0 1], with smaller numbers meaning less absorption)

beer's law derivation

$$dI = -CIdx$$

$$\frac{1}{I}dI = -Cdx$$

$$\int \frac{1}{I}dI = \int -Cdx$$

$$\ln I = -Cx + C'$$

$$I(x) = e^{-Cx+C'}$$

$$I(x) = e^{C'}e^{-Cx}$$

$$I(x) = ke^{-Cx}$$

beer's law derivation

$$I(x) = ke^{-Cx}$$

$$I(0) = I_o$$

$$I(x) = I_0 e^{-Cx}$$

$$I(1) = aI(0)$$

a is an attentuation constant (0.<a<= 1.0)

$$I_0 a = I_0 e^{-C}$$

$$a = e^{-C}$$

$$\ln a = -C$$

$$I(s) = I_0 e^{\ln(a)\ s}$$

as distance s (into the material) increases, ***I*** decreases

some notes about refraction

we have approximated the Fresnel equations and Beer's law - these approximations apply to dielectrics - materials that are nonconductive

note: in the real world, the index of refraction varies with wavelength - this effect is ignored in all production renderers

recursive ray tracing architecture

your loop for rendering the scene now loops over:

- each pixel
- each object (for intersection)
- each light (for shading)
  - each object (for shadow-ray intersection)

it's getting complicated!

```
rgb L (ray, min_t, max_t, recurse_limit, ray_type, light,...)
{
}
```

```
rgb L (ray, min_t, max_t, recurse_limit, ray_type, light,...)
{
  if (recurse_limit == 0) return [0 0 0]

  if (ray_type == SHADOW_RAY) {
  …
  }

  get closest intersection w/ scene, along ray
  if there is no intersection, return [0 0 0]

  for each light "thisLight":
     compute specular + diffuse shading and add into R[] color accumulator vector


  add in ambient if needed

  if (material is NOT reflective) return R[]
  else {
    compute reflected ray "ref_ray" and call L recursively
 }
}
```

```
rgb L (ray, min_t, max_t, recurse_limit, ray_type, light,...)
{
  if (recurse_limit == 0) return [0 0 0]

  if (ray_type == SHADOW_RAY) {
    if any intersection on [min_t, max_t] interval,
      return [0 0 0]
    else return light->spd()
  }

  get closest intersection w/ scene, along ray
  if there is no intersection, return [0 0 0]
  for each light "thisLight":
    compute ray "s_ray", frm intrsctn point to light, & maximum t-value "s_max_t"
    L_rgb = L(s_ray, 0.0001, s_max_t, 1, SHADOW_RAY, thisLight,...)
    if (L_rgb.length() > 0) {
      compute specular + diffuse shading and add into R[] color accumulator vector
    }
  }
  add in ambient if needed
  if (material is NOT reflective) return R[]
  else {
    compute reflected ray "ref_ray"
    return (R[] + Kr⊗L (ref_ray, .0001, +infinity, recurselimit - 1, REGULAR_RAY, ...)
  }
}
```

recursive ray tracing architecture

(note: ⊗ means component-wise vector multiply)

Now, in your per-pixel nested loops, just set the pixel's value to be the result of :

L (camera_ray,...)

assignment: theme 1, milestone 3

- truly reflective materials
- ambient lights
- wavefront (obj) file read - more triangles!

- and: handle two-sided lighting & shading with yellow diffuse material (see HW online)