CMSC 435
Introductory Computer Graphics
Basic Ray

Penny Rheingans
UMBC

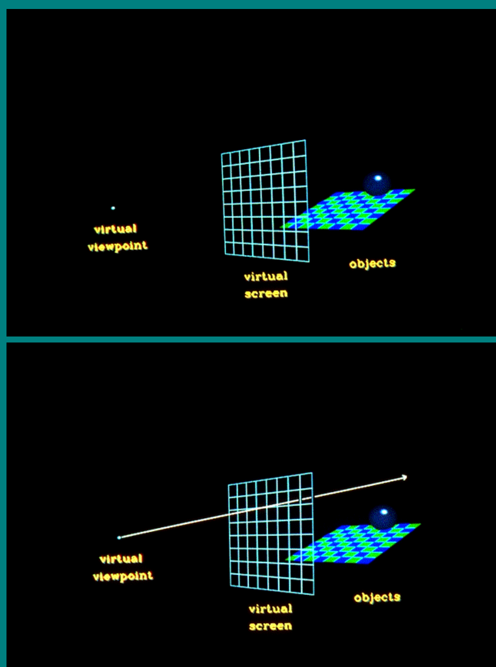
Announcements

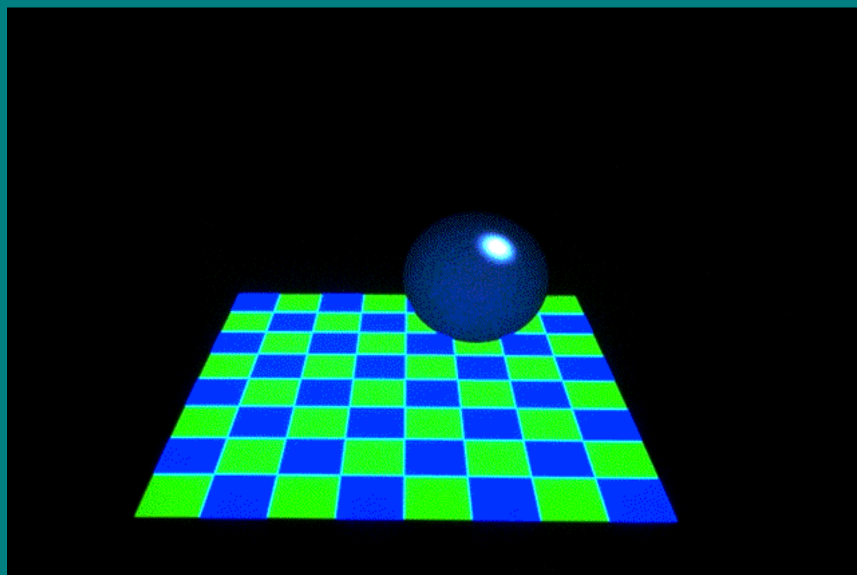
Visibility Problem

- Rendering: converting a model to an image
- Visibility: deciding which objects (or parts) will appear in the image
  - Object-order
  - Image-order

Raytracing

- Given
  - Scene
  - Viewpoint
  - Viewplane
- Cast ray from viewpoint through pixels into scene



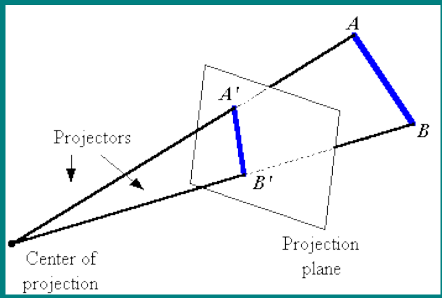


Raytracing Algorithm

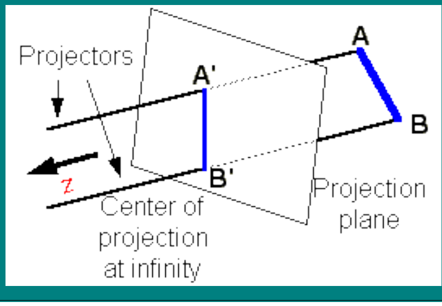
```
Given
    List of polygons { P1, P2, ..., Pn }
    An array of intensity [ x, y ]
{
   For each pixel (x, y) {
      form a ray R in object space through the
        camera position C and the pixel (x, y)
      Intensity [ x, y ] = trace ( R )
      }
   Display array Intensity
}
```

Projection

- Perspective
  - Line AB projects to A'B' (perspective projection)



- Parallel
  - Line AB projects to A'B' (parallel projection)
  - Projectors AA' and BB' are parallel



Simple Parallel Tform

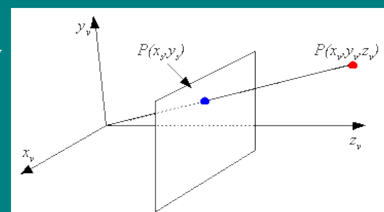
View plane is normal to direction of projection

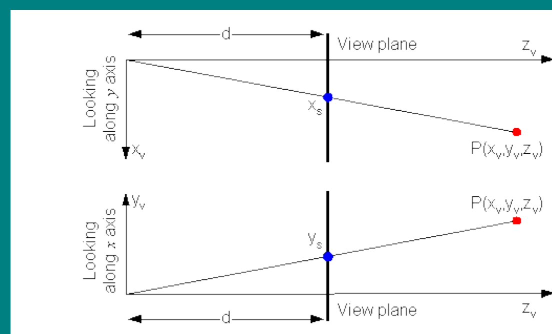
$x_s = x_v,\ y_s = y_v,\ z_s = 0$

$$T_{ort} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Simple Perspective Tform

- Assume line from center of projection to center of view plane parallel to view plane normal.
- Center of projection is at origin.
- Have P(xv, yv, zv)
- Want P(xs, ys)



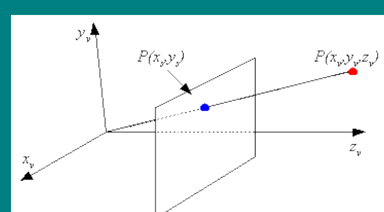


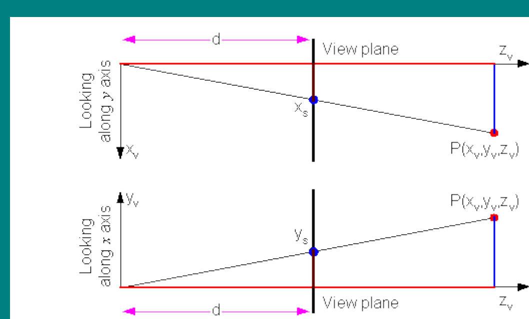
Simple Perspective Tform

- Have P(xv, yv, zv)
- Want P(xs, ys)
- By similar triangles:

$$\frac{x_s}{d} = \frac{x_v}{z_v}, \frac{y_s}{d} = \frac{y_v}{z_v}$$

$$\Rightarrow x_s = \frac{x_v}{z_v/d}, y_s = \frac{y_v}{z_v/d}$$

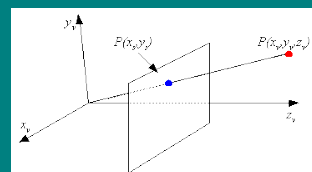




Simple Perspective Tform

- Have $P(x_v, y_v, z_v)$, want $P(x_s, y_s)$
- By similar triangles:

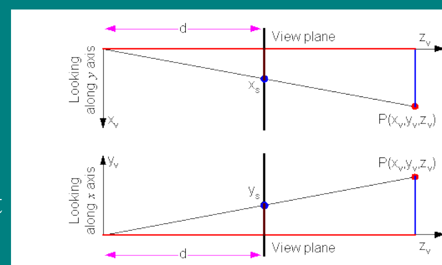
$$\frac{x_s}{d} = \frac{x_v}{z_v}, \frac{y_s}{d} = \frac{y_v}{z_v} \Rightarrow x_s = \frac{x_v}{z_v/d}, y_s = \frac{y_v}{z_v/d}$$



- In homogeneous coords

$x = x_v$, $y=y_v$, $z = z_v$, $w = z_v/d$

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x_v \\ y_v \\ z_v \\ 1 \end{bmatrix}$$



- Do perspective divide to get screen coords

$x_s = x/w$, $y_s = y/w$, $z_s = z/w = d$

Raytracing Algorithm

```
Given
    List of polygons { P₁, P₂, ..., Pₙ }
    An array of intensity [ x, y ]
{
   For each pixel (x, y) {
      form a ray R in object space through the
        camera position C and the pixel (x, y)
      Intensity [ x, y ] = trace ( R )
      }
   Display array Intensity
}
```

Raytracing Algorithm

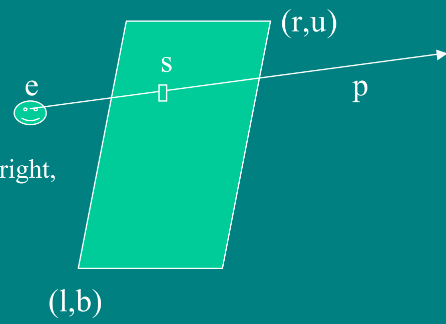
```
intensity Function  trace ( Ray )
{
   for each polygon P in the scene
      calculate the intersection of P and the ray R
   if ( The ray R hits no polygon )
        return ( background intensity )
   else {
      find the polygon P with the closest
        intersection
      calculate intensity I at intersection point
      return ( I ) // more to come here later
    }
}
```

Raytracing Algorithm

```
intensity Function  trace ( Ray )
{
   calculate the intersection of nearest polygon P
     and the ray R
   if ( The ray R hits no polygon )
        return ( background intensity )
   else {
      find the polygon P with the closest
        intersection
      calculate intensity I at intersection point
      return ( Illuminate( I, trace(reflect ),
        trace( refract ) ) )
    }
}
```

Computing Viewing Rays

- Parametric ray
  - $\mathbf{p}(t) = \mathbf{e} + t(\mathbf{s} - \mathbf{e})$
- Camera frame
  - E: eye point
  - u,v,w: basis vectors pointing right, up, backward
- Screen position
  - orthographic
    - $u_s = l + (r-l)(i+0.5)/n_x$
    - $v_s = b + (u-b)(j+0.5)/n_y$
    - $\mathbf{s} = (\mathbf{e} + u_s\mathbf{u} + v_s\mathbf{v}) - \mathbf{w}$
  - Perspective
    - $u_s = l + (r-l)(i+0.5)/n_x$
    - $v_s = b + (u-b)(j+0.5)/n_y$
    - $\mathbf{s} = (\mathbf{e}) + u_s\mathbf{u} + v_s\mathbf{v} - d\mathbf{w}$



Calculating Intersections

- Define ray parametrically:

  $x = x_0 + t(x_1 - x_0) = x_0 + t\Delta x$

  $y = y_0 + t(y_1 - y_0) = y_0 + t\Delta y$

  $z = z_0 + t(z_1 - z_0) = z_0 + t\Delta z$
- If (x0, y0, z0) is center of projection and (x1, y1, z1) is center of pixel, then

  $0 <= t <= 1$: points between those locations

  $t < 0$: points behind viewer

  $t > 1$: points beyond view window

Ray-Sphere Intersection

- Sphere in vector form
  - $(\mathbf{p}-\mathbf{c})\bullet(\mathbf{p}-\mathbf{c})-R^2=0$
- Ray
  - $\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}$
- Intersection with implicit surface f(t) when
  - $f(\mathbf{p}(t)) = 0$
  - $(\mathbf{e}+t(\mathbf{s}-\mathbf{e})-\mathbf{c})\bullet(\mathbf{e}+t(\mathbf{s}-\mathbf{e})-\mathbf{c})-R^2 = 0$
  - $(\mathbf{d}\bullet\mathbf{d})t^2+2\mathbf{d}\bullet(\mathbf{e}-\mathbf{c})t+(\mathbf{e}-\mathbf{c})\bullet(\mathbf{e}-\mathbf{c})-R^2 = 0$
  - $t=(-\mathbf{d}\bullet(\mathbf{e}-\mathbf{c})\pm\text{sqrt}((\mathbf{d}\bullet(\mathbf{e}-\mathbf{c}))^2-(\mathbf{d}\bullet\mathbf{d})((\mathbf{e}-\mathbf{c})\bullet(\mathbf{e}-\mathbf{c})-R^2)/(\mathbf{d}\bullet\mathbf{d})$
- Normal at intersection p
  - $n=(p-c)/R$

Calculating Intersections: Pgons

- Given ray and polygon:

$x = x_0 + t(x_1 - x_0) = x_0 + t\Delta x$

$y = y_0 + t(y_1 - y_0) = y_0 + t\Delta y$

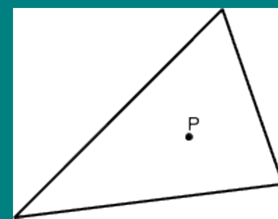
$z = z_0 + t(z_1 - z_0) = z_0 + t\Delta z$

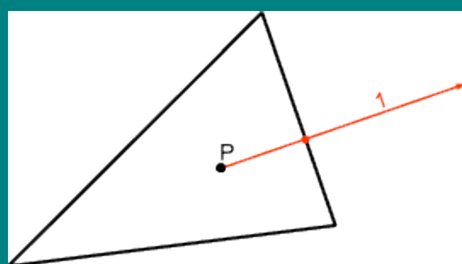
Plane : $Ax + By + Cz + D = 0$

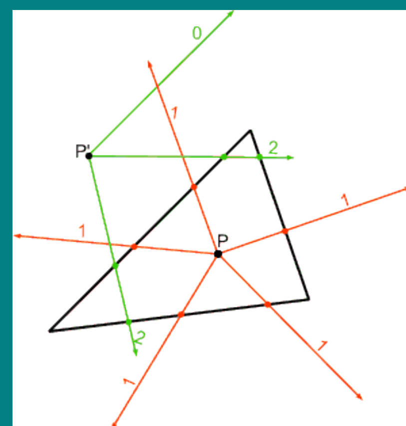
1. What is intersection of ray and plane containing pgon?
   - Substituting for x, y, z:
   - $A(x_0 + t\Delta x) + B(y_0 + t\Delta y) + C(z_0 + t\Delta z) + D = 0$
   - $t(A\Delta x + B\Delta y + C\Delta z) + (Ax_0 + By_0 + Cz_0 + D) = 0$
   - $t = -(Ax_0 + By_0 + Cz_0 + D) / (A\Delta x + B\Delta y + C\Delta z)$
2. Does ray/plane intersection point lie in polygon?
   - Substitute into line equations for pgon edges: does point lie inside all edges? (only works for convex)
   - Count edge crossings of ray from point to infinity

Point in Polygon?

- Is P in polygon?
- Cast ray from P to infinity
  - One crossing -> inside
  - Zero, Two crossings -> outside

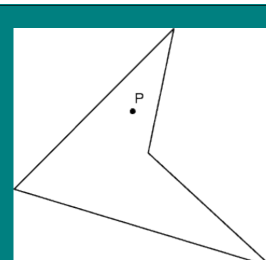


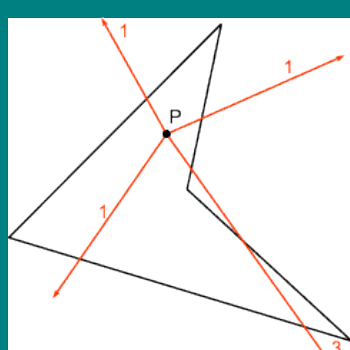


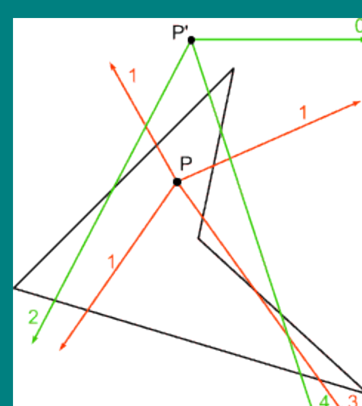


Point in Polygon?

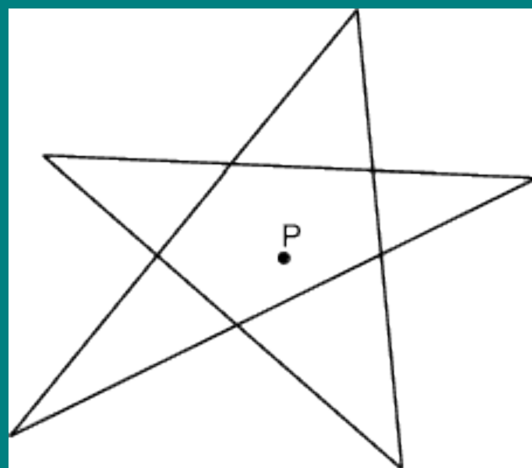
- Is P in concave polygon?
- Cast ray from P to infinity
  - Odd crossings -> inside
  - Even crossings -> outside







What happens?



Ray-Triangle Intersection

- Intersection of ray with barycentric triangle
  - $\mathbf{e}+t\mathbf{d} = \mathbf{a}+\beta(\mathbf{b}-\mathbf{a})+\gamma(\mathbf{c}-\mathbf{a})$
  - In triangle if $\beta > 0$, $\gamma > 0$, $\beta + \gamma < 1$

```
boolean raytri (ray r, vector a, vector b, vector c,
   interval [t0,t1] ) {
    compute t
    if (( t < t0 ) or (t > t1))
         return ( false )
    compute γ
    if ((γ < 0 ) or (γ > 1))
         return ( false )
    compute β
    if ((β < 0 ) or (β > 1))
         return ( false )
    return true
}
```

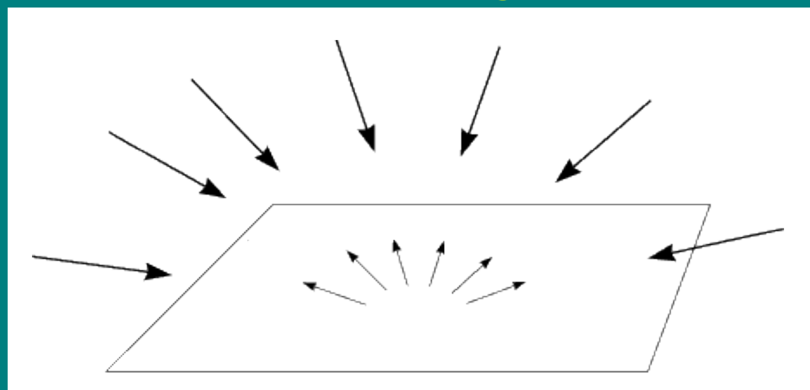
Raytracing Characteristics

- Good
  - Simple to implement
  - Minimal memory required
  - Easy to extend
- Bad
  - Aliasing
  - Computationally intensive
    - Intersections expensive (75-90% of rendering time)
    - Lots of rays

Basic Concepts

- Terms
  - Illumination: calculating light intensity at a point (object space; equation) based loosely on physical laws
  - Shading: algorithm for calculating intensities at pixels (image space; algorithm)
- Objects
  - Light sources: light-emitting
  - Other objects: light-reflecting
- Light sources
  - Point (special case: at infinity)
  - distributed

Ambient light



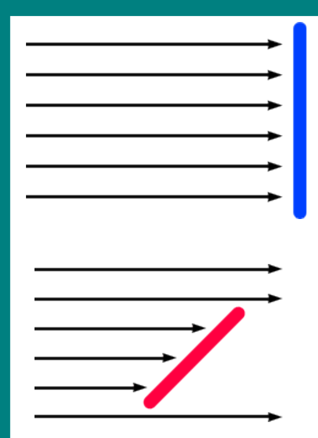
I_a = intensity of ambient light

K_a = reflection coefficient

$I = k_a I_a$ = reflected intensity

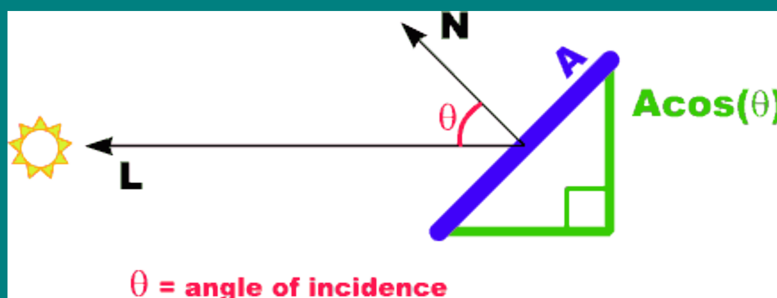
Lambert's Law

- Intensity of reflected light related to orientation



Lambert's Law

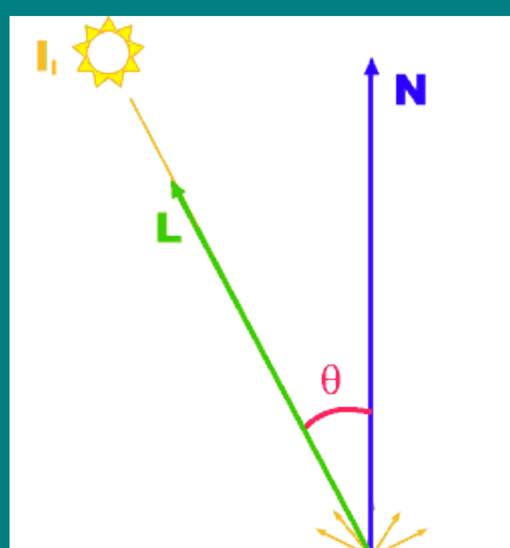
- Specifically: the radiant energy from any small surface area dA in any direction θ relative to the surface normal is proportional to cos θ



Diffuse Reflection

$I_{diff} = k_d I_l \cos\theta$

$= k_d I_l \, (N \bullet L)$



Combined Model

$I_{total} = I_{amb} + I_{diff}$

$= k_a I_a + k_d I_l (N \bullet L)$

Adding color:

$I_R = k_a I_{aR} O_{dR} + k_d I_{lR} O_{dR} (N \bullet L)$

$I_G = k_a I_{aG} O_{dG} + k_d I_{lG} O_{dG} (N \bullet L)$

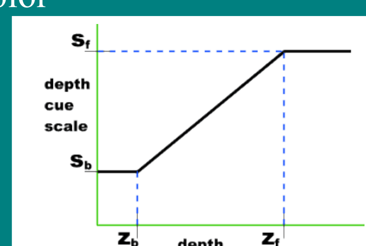
$I_B = k_a I_{aB} O_{dB} + k_d I_{lB} O_{dB} (N \bullet L)$

For any wavelength λ:

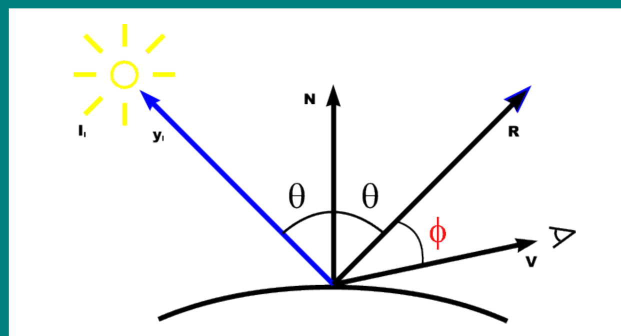
$I_\lambda = k_a I_{a\lambda} O_{d\lambda} + k_d I_{l\lambda} O_{d\lambda} (N \bullet L)$

Adding Attenuation

- Attenuation of light source due to distance
  - $F_{att} = 1/d_L^2$ or $\min(1/(C_1+C_2 d_L+C_3 d_L^2),1)$
    - where d_L is distance to the light
  - Behavior of $1/d_L^2$
    - Far from light: little change
    - Near light: much change
    - Accurate, but looks wrong
- Atmospheric attenuation of color
  - $I_\lambda' = S_O I_\lambda + (1-S_O) I_{dc\lambda}$
    - where $I_{dc\lambda}$ is the depth cue color
    - $S_O = S_b + (Z_O - Z_b)(S_f - S_b)/(Z_f - Z_b)$



Specular Reflection



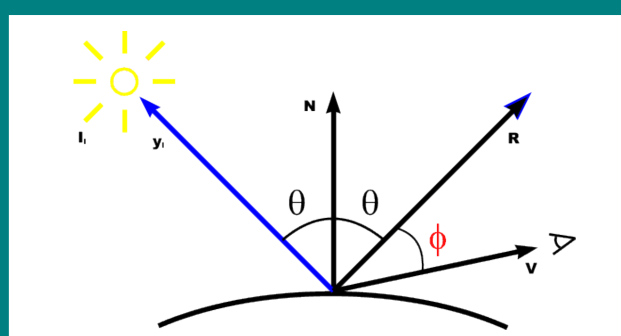
For specific wavelength λ

$I_{spec\lambda} = k_{s\lambda} I_{\lambda} \cos^n \phi$

$= k_{s\lambda} I_{\lambda} (R \cdot V)^n$

Not dependent on surface color → white highlights

Specular Reflection



For specific wavelength λ

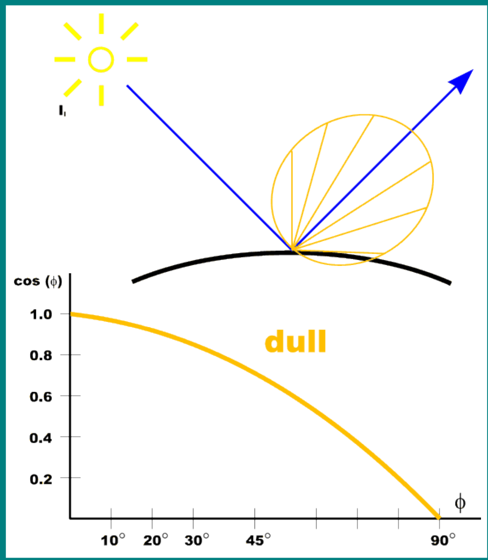
$I_{spec\lambda} = k_{s\lambda} I_{\lambda}\, O_{s\lambda} \cos^n \phi$

$= k_{s\lambda} I_{\lambda}\, O_{s\lambda}\, (R \cdot V)^n$

Dependent on surface color → colored highlights

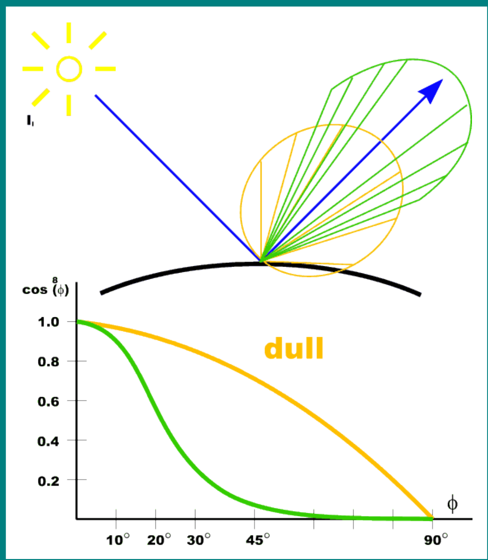
Specular Reflection

- Dull highlights
  - Gradual falloff
  - Approximated by cos ϕ



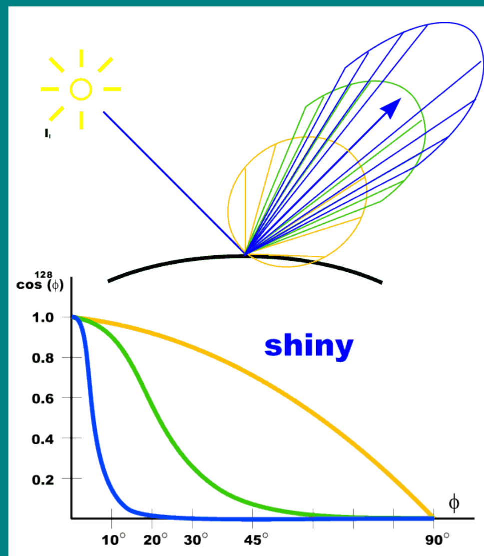
Specular Reflection

- Glossy highlights
  - Steeper falloff
  - Approximated by $\cos^8\phi$



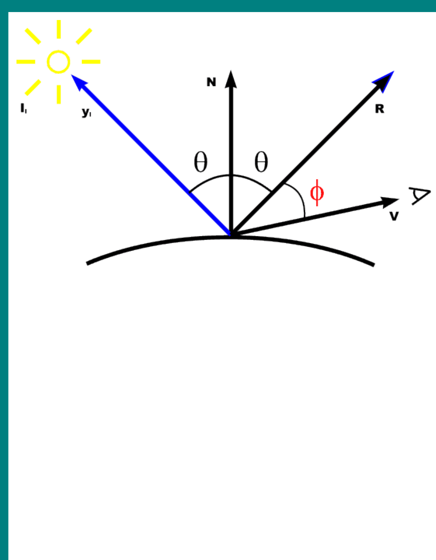
Specular Reflection

- Shiny highlights
  - Steep falloff
  - Approximated by $\cos^{128}\phi$



Calculating the Reflection Vector

- Specular:
  $I_{spec\lambda} = k_{s\lambda} I_\lambda O_{s\lambda} (R \bullet V)^n$
- Have L, want R



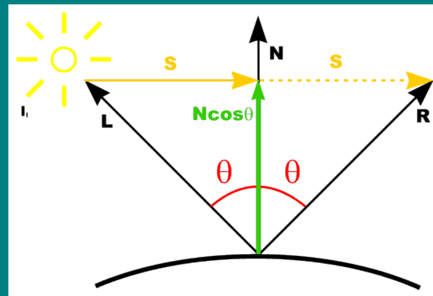
Calculating the Reflection Vector

- Mirror L about N

  $R = N\cos\theta + S$

  $\quad = 2N\cos\theta - L$

  $\quad = 2N(N \bullet L) - L$



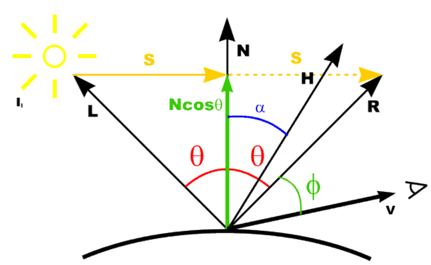
Where N, L are unit length
Projection of L on N is N cos θ
S = N cos θ -L

- $I_{spec\lambda} = k_{s\lambda} I_\lambda O_{s\lambda} (R \bullet V)^n$

  $\quad = k_{s\lambda} I_\lambda O_{s\lambda} (2N(N \bullet L) - L \bullet V)^n$

Calculating the Reflection Vector

- Alternatively: use halfway vector H
  - $H = (L+V)/|L+V|$
- Maximum highlight when H=N (because then R=V)
  - $I_{spec} = k_s I_l (H \bullet N)^n$
  - $H \bullet N = \cos\alpha$
- Two methods can give different results $\alpha \neq \phi$



Combined Model

$I_{total} = I_{amb} + I_{diff} + I_{spec}$

$= k_a I_a + k_d I_l \, (N \bullet L) + k_s I_l \, (N \bullet H)^n$

Multiple lights:

$= k_a I_a + \sum (k_d I_{li} \, (N \bullet L) + k_s I_{li} \, (N \bullet H)^n)$

By wavelength (white highlights):

$= k_a I_a \, O_{d\,\lambda} + \sum (k_d I_{li} \, (N \bullet L) \, O_{d\,\lambda} + k_s I_{li} \, (N \bullet H)^n)$

By wavelength (colored highlights):

$= k_a I_a \, O_{d\,\lambda} + \sum (k_d I_{li} \, (N \bullet L) \, O_{d\,\lambda} + k_s I_{li} \, (N \bullet H)^n \, O_{s\lambda})$

By wavelength (more metallic highlights):

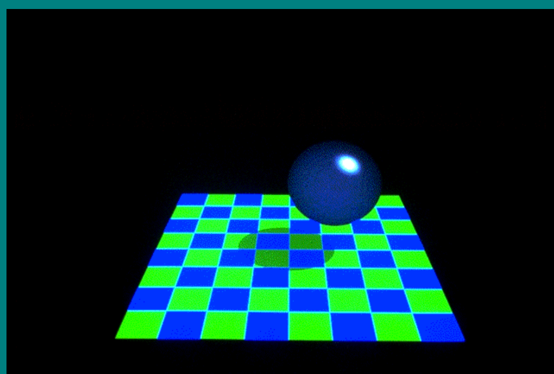
$= k_a I_a \, O_{d\,\lambda} + \sum (k_d I_{li} \, (N \bullet L) \, O_{d\,\lambda} + k_s(\lambda,\theta) I_{li} \, (N \bullet H)^n \, O_{s\lambda})$

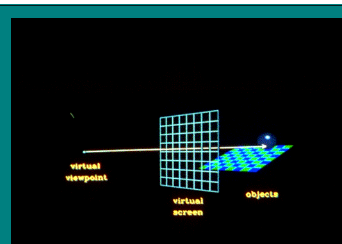
Basic Raytracing Program

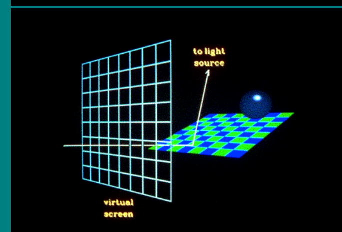
```
{
   for each pixel (x, y) do {
      compute viewing ray
      if (ray hit an object with t>0) then {
        compute n
        evaluate shading model
        set pixel to that color
        }
      else
        set pixel color to background color
}
```

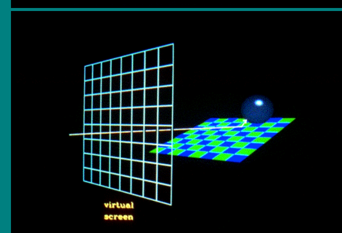
Effects

- Visibility
- Illumination
- Shadow







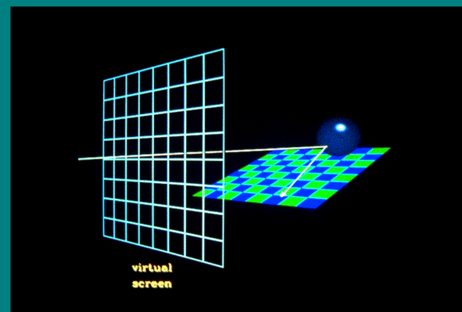


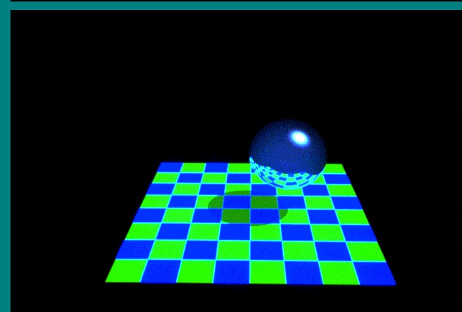
Shadow Algorithm

```
Function raycolor(ray e+td, real t_0, real t_1)
{
   hit-record rec, srec
   if (scene->hit(e+td, t_0, t_1) then {
      p = e+(rec.t)d
      color c = rec.k_a*I_a
      if (not scene->hit(p+sl, ε, ∞, srec) then {
         vector h = unit(unit(l)+unit(-d))
         c = c + rec.k_d*I*max(0,rec.n•l) +
           rec.k_s*I(rec.n•h)^(rec.p)
         }
      return c
      }
   else
     return background color
}
```

Effects

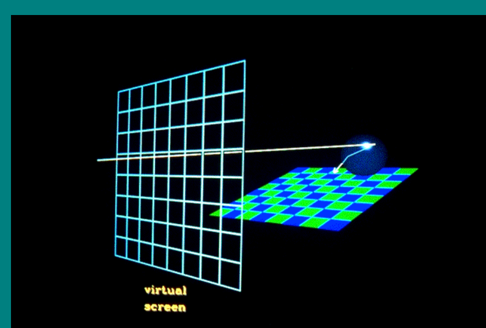
- Reflection
- Calculate ray direction
  - $r = d-2(d \cdot n)n$
  - d points from eye to surface
- Trace ray
  - $m = \text{raycolor}(p+sr, \varepsilon, \infty)$
- Composite
  - $c = c + k_m m$

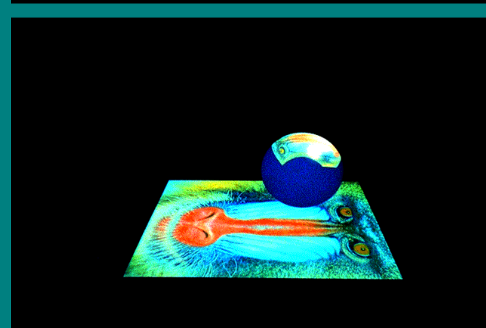




Effects

- Refraction





How many bounces?