Mirror Puzzles

- My image in the mirror has his head up, so mirror reflection does not invert up and down. Yet, his watch is on his right wrist, so mirror reflection inverts left and right. Why?

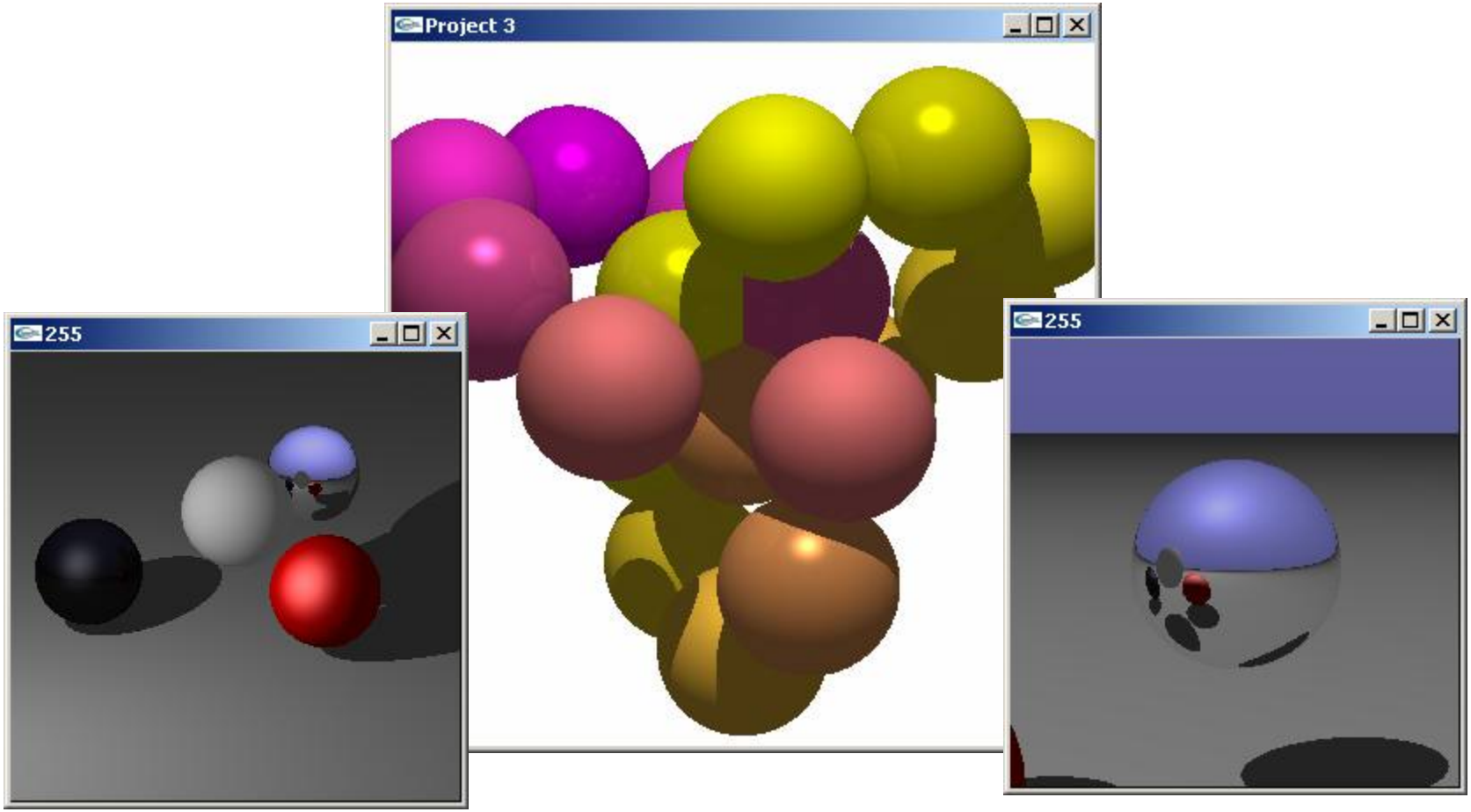
- There are two tall adjacent vertical mirror at 90 degrees from each other in the corner of a room. Where do you have to stand to see yourself? Prove it? How do you appear?

- Now we have 3 small mirrors at 90 degrees from each other in the top corner of a room. One is glued to the ceiling. The other two to the walls. Each mirror touches the other two along an edge. From where can I see myself? How do I look? What are such contraptions used for?

Photorealism

Georgia Tech College of Computing
Interactive & Intelligent Computing

- View setup
- Shooting/tracing rays



Lecture Objectives

- Know how to design the overall structure and the details of a simple ray-tracer on triangle meshes.
  - How to define the view parameters
  - How to test for ray-triangle intersection
  - How to test triangle orientation
  - How to select the front-most triangle
  - How to check visibility from light sources
  - How to compute the reflected color using a simple reflection model
  - How to deal with mirrors and highly reflective surfaces
- Know how to extend it to CSG models with quadric and triangle-mesh primitives.
- Understand the ray-tracing cost/limitation tradeoff and the principle of radiosity

Set-up the view and compute rays

- **Physical parameters of the real screen/viewer**
  - Assume viewpoint E to project onto the center O of screen
  - What parameters define the pixels?
  - Assume pixel-size unit
- **Specify scale/position/size of the virtual screen/viewer**
  - Input: Scale *s*, screen-center O, viewpoint E, resolution (2w+1)x(2w+1)
  - How would you compute screen coordinate vectors I, J, K?
    - Assume no "roll" (sideway tilt of the head).
    - Let **U** be the vertical up-vector.
    - Let **I** be horizontal

I:=VO×U; J:=I×VO; I:=I/||I||; J:=J/||J||;

- **Build all the rays**
  - **Given** O, E, s, w
  - **write algorithm to enumerate all the pixels,** P**, as 3D points**
    - On the virtual screen
  - **Each ray Ray**(E,P) **will be defined by** E**, and** P
    - Semi infinite line segment starting at **E** and containing **P**

Overall ray-tracing algorithm

```
I:=VO×U; I:=I/||I||;                    # horizontal screen axis
J:=I×VO; J:=J/||J||;                    # vertical screen axis
FOR x=-w TO w DO
    FOR y=-w TO w DO {
        P:=O+sxI+syJ;                   # pixel as 3D point
        color[x,y]:=HitColor(E,P)}      # set pixel color
```

HitColor(E,P) returns the color reflected by the first surface point that this ray hits

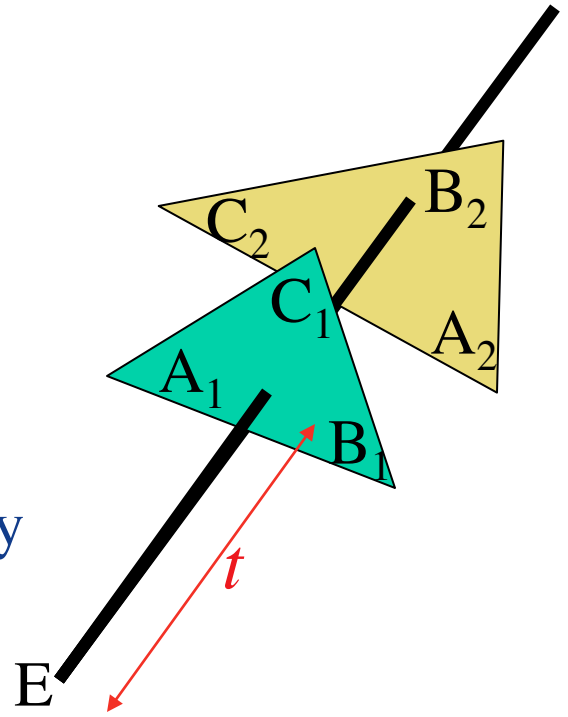
Hidden surface removal

Given several front triangles intersected by Ray(**E**,**P**)
how would you select the visible one?

The one with the smallest distance *t*

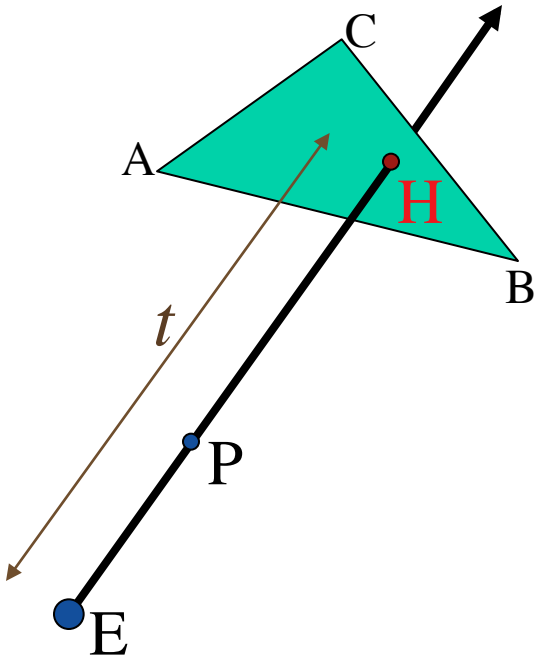
Need only consider front-facing triangles

Needs only consider triangles hit by the ray



HitColor(E,P): compute the hit point

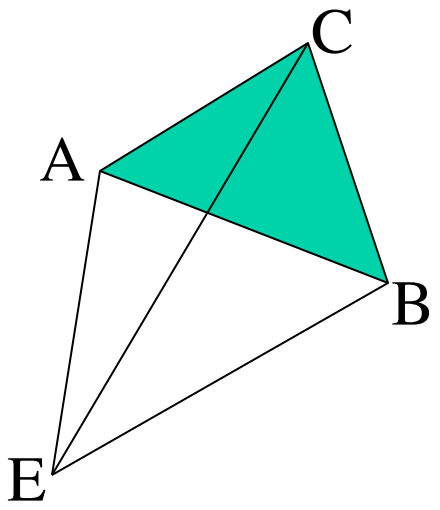
```
Proc HitColor(E,P) {
z:=∞;
FOREACH (Triangle Tri(A,B,C) ) {
  if (IsFrontFacing(A,B,C,E) && RayHitsTri(E,P,A,B,C)){
    t:=DistAlongRay(E,P,A,B,C);
    if (t<z) {z:=t;  N:=AB×AC; N.makeUnit();};
if (z==∞)  {return(backGround);}
else {return( ReflectedIntensity (E, E + t EP, N, L) );};
}
```



E + *t* EP *is the first surface point H hit by the ray*

IsFrontFacing(**A**,**B**,**C**,**E**): back face culling

Given an oriented triangle Tri(A,B,C) and a viewpoint E, how would you test whether the triangle is front facing?



Proc s(**A**,**B**,**C**,**D**) {RETURN (AB×AC)•AD>0}

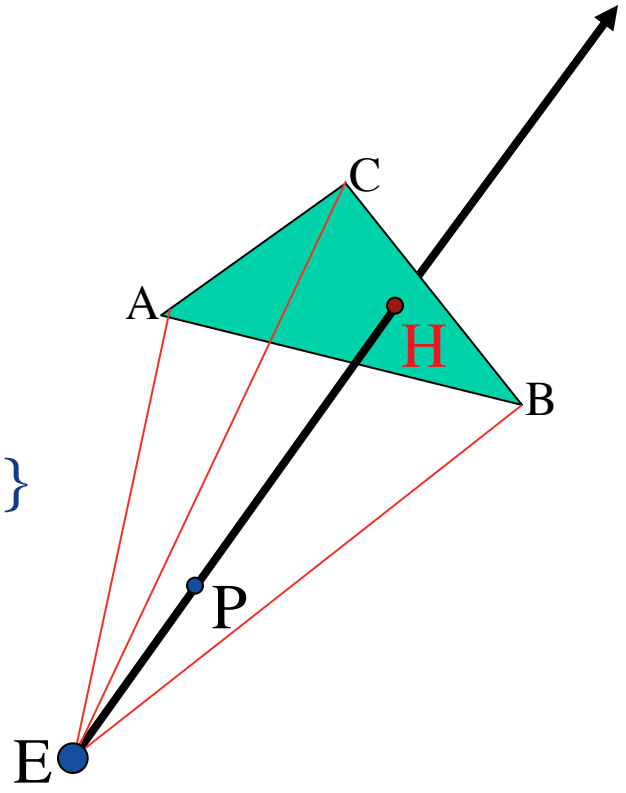
Proc IsFrontFacing(**A**,**B**,**C**,**E**) {RETURN s(**A**,**B**,**C**,**E**) }

RayHitsTri(**E,P,A,B,C**): ray/triangle hit test

Given a front triangle Tri(A,B,C) and a viewpoint E, and a pixel P, how to best test whether Ray(E,P) intersects Tri(A,B,C)?

Proc RayHitsTri(**E**,**P**,**A**,**B**,**C**)

{RETURN (*s*(**P**,**B**,**C**,**E**) AND
s(**A**,**P**,**C**,**E**) AND
s(**A**,**B**,**P**,**E**))}



DistAlongRay(**E,P,A,B,C**): computes depth and **H**

Given a front triangle Tri(A,B,C) and a viewpoint E, and a pixel P, how to compute the distance along Ray(E,P) between E and the plane that contains Tri(A,B,C)?

(AB×AC)•AH= 0 AND **H=E+*t*EP**

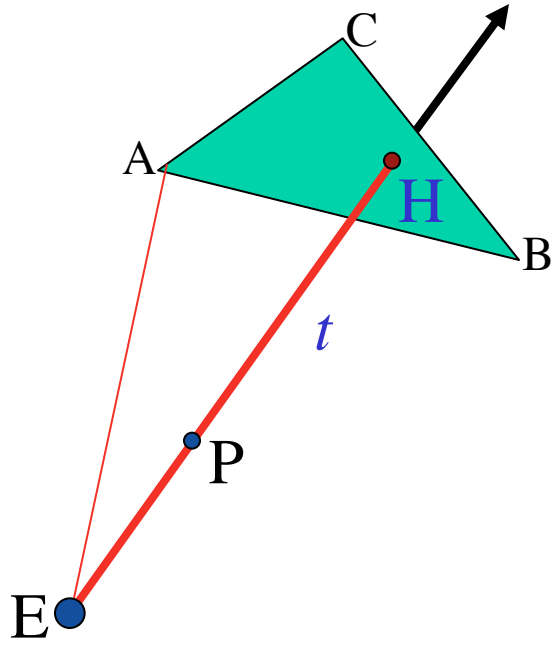
*(same unit ||**EP**|| for all triangles)*

Solve for t

(AB×AC)•(AE+*t* EP) = 0

(AB×AC)•AE= - *t* (AB×AC)•EP

$$t = (\mathbf{AB}\times\mathbf{AC})\bullet\mathbf{EA} \,/\, (\mathbf{AB}\times\mathbf{AC})\bullet\mathbf{EP}$$



Proc DistAlongRay(**E,P,A,B,C**)

{ return ((**AB×AC)•EA / (AB×AC)•EP**)}

Reflected Light: ReflectedIntensity (E,H,N,L)

Compute intensity emitted from an infinite light situated in the direction L and reflected triangle with normal N towards E

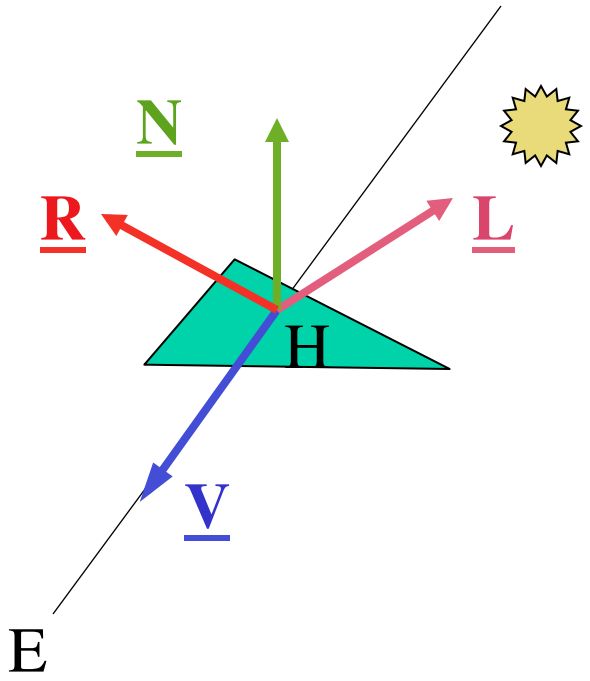
View direction V:= HE/ ||HE|| ;

Reflected light direction R:= 2(N•L)N–L ;

Convex weighted sum of three components

(a+d+s = 1?)

- Ambient: constant A
- Diffuse: N•L
- Specular: $(R \bullet V)^k$



Proc ReflectedIntensity (**E**, **H**, **N**, **L**) {

V:= **HE**/ ||**HE**|| ;

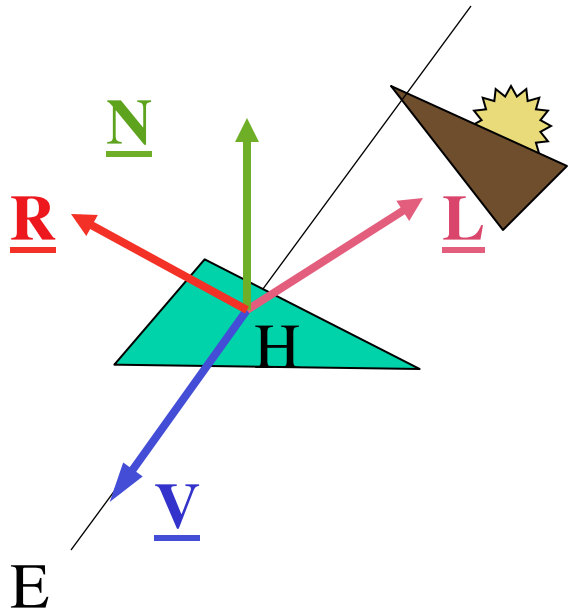
R:= 2(**N**•**L**)**N**-**L** ;

return(aA+ d**N**•**L**+ s(**R**•**V**)^k) }

Shadows

For each point H seen from E, add the diffuse and specular components only when the ray from H in the direction L does not hit any triangle.

Use Lit (H,L) to test



Proc ReflectedIntensity (**E**, **H**, **N**, **L**) {

V:= **HE**/ ||**HE**|| ;

R:= 2(**N**•**L**)**N**–**L** ;

col:=aA;

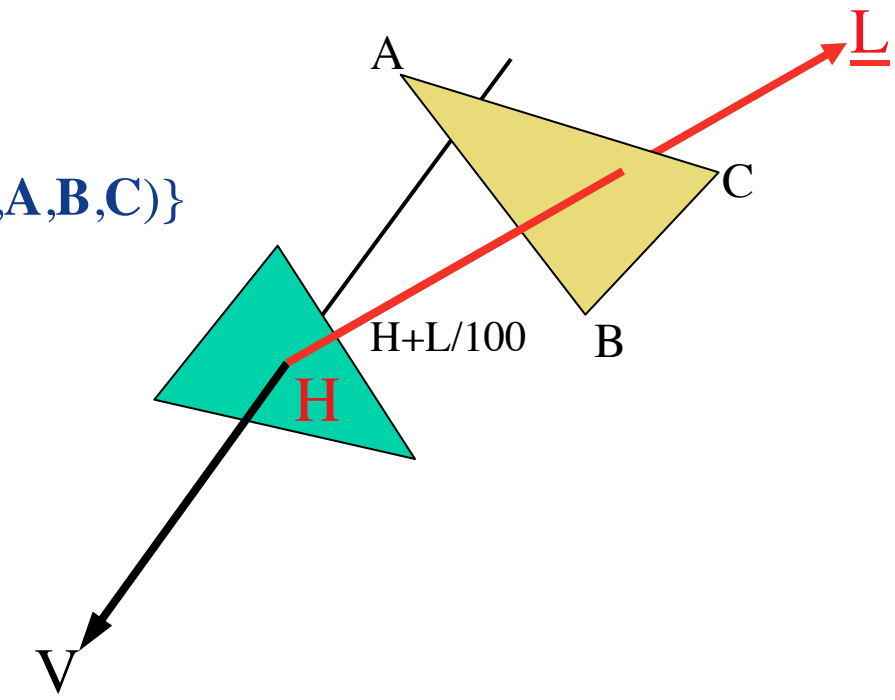
if (Lit (**H**,**L**)) {**col+=** d**N**•**L**+ s(**R**•**V**)^k;};

return(col); }

Lit (**H**,**L**) : Does H see the light?

Check whether the point H on Tri(A,B,C) is seen from the light in the direction L

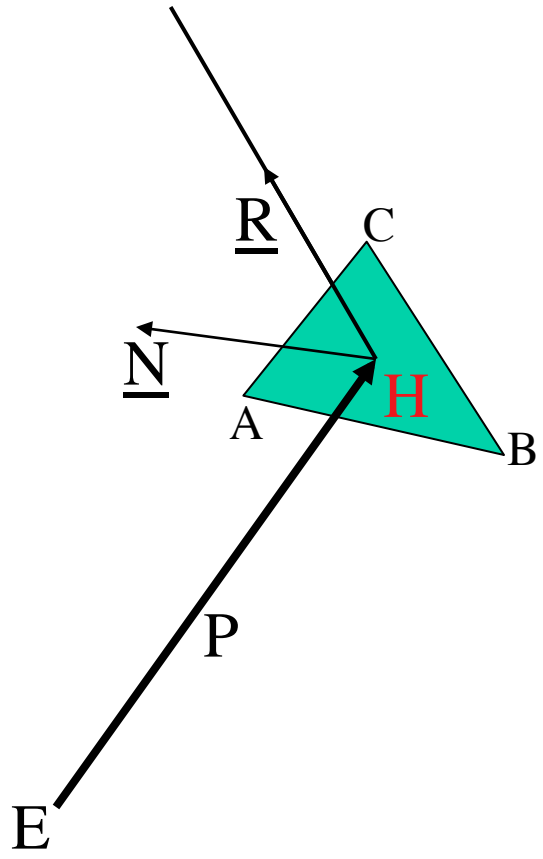
```
Proc Lit (H,L) {
    foreach (Triangle Tri(A,B,C) ) {
        if (IsFrontFacing(A,B,C,H)
            && RayHitsTri(H,H+L/100,A,B,C)}
            { return(false);};};
    return(true); }
```



Mirror reflections

Suppose that Tri(A,B,C), which is the triangle visible from E through pixel P, is a mirror. What should we do?

Cast Ray(H,R)… recursively



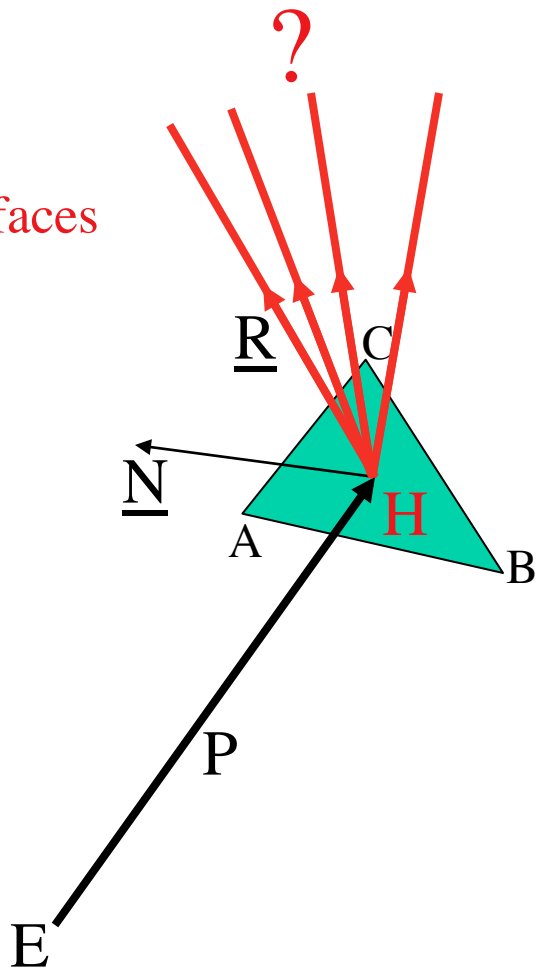
What's wrong with this picture?

- Visibility? OK
- Shadows? OK, but, single light at **infinity**
- Color? No light reflection off other surfaces
- How to fix it?

Cast many **secondary** rays from H.

Test what the secondary rays hit

–If they hit a triangle, compute the first hit
–Check if that new hit point is in the shadow
–Compute the light reflected by that point
–Cast new rays from it….



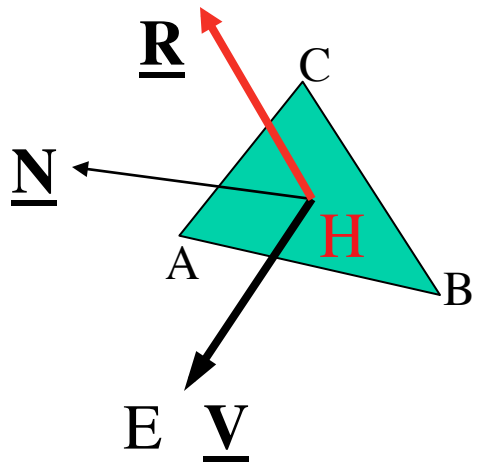
Limited Ray-tracing

- If you had to cast a single secondary ray, what would it be?

The ray from **H** towards the reflection **R** of **V** around **N**?

Why?

$$\mathbf{R} := 2(\mathbf{N} \bullet \mathbf{V})\mathbf{N} - \mathbf{V} \; ;$$



CSG

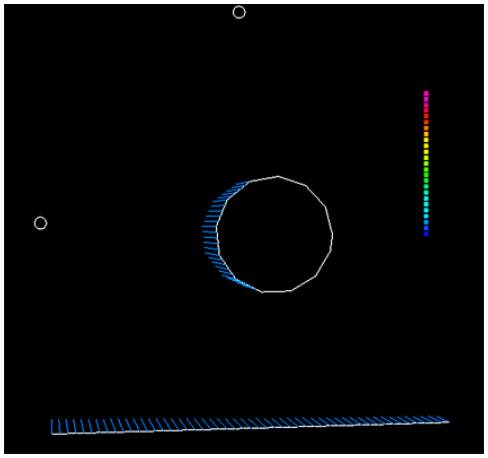
- Add a sphere to your scene
  - What needs to be changed in the algorithm?
  - How do you compute the ray-sphere intersection?

    Solve $\mathbf{CH} \bullet \mathbf{CH} = r^2$ for t with $\mathbf{H} = \mathbf{E} + t\,\mathbf{EP}$

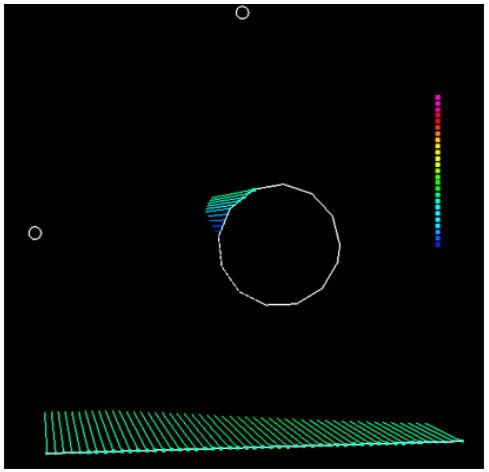
- Add a CSG combination of spheres
  - What needs to be changed in the algorithm?
  - How do you compute the intersection of a ray with a CSG object?

- Add a CSG combination of solids bounded by water-tight T-meshes
  - What needs to be changed in the algorithm?

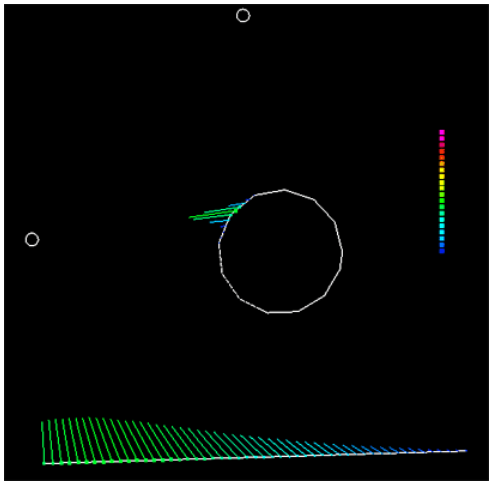
2D simulation (Ingram)



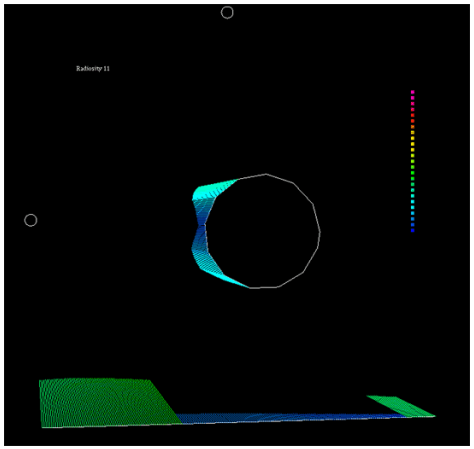
Ambient



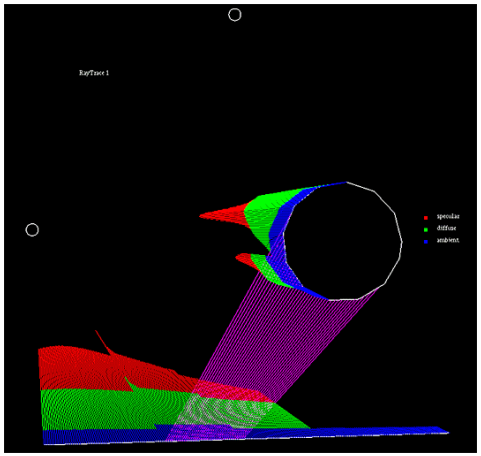
Diffuse



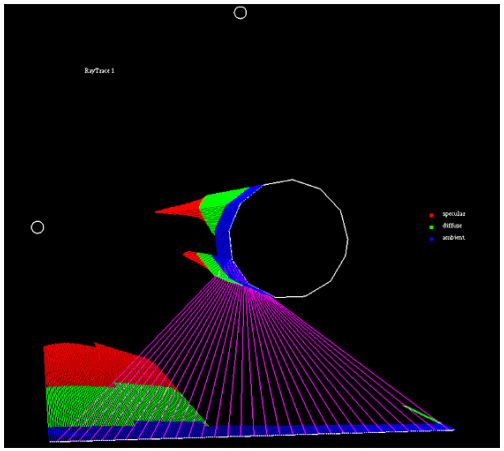
Specular



Radiosity



Sphere to board



Board to sphere

Real or synthetic? Justify!