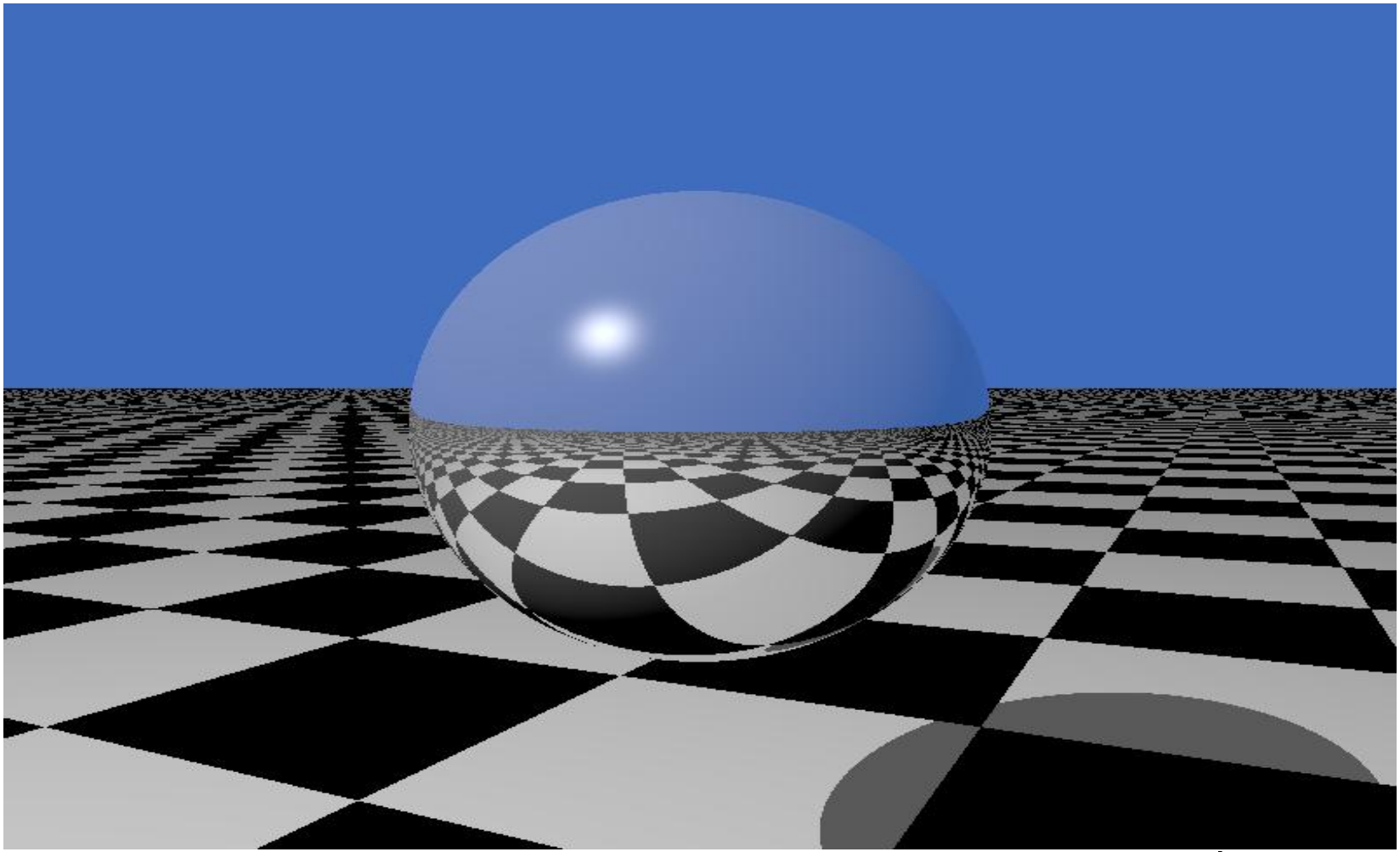
lecture 18

- ray tracing
- environment mapping
- refraction

Recall Ray Casting (lectures 7, 8)

```
for each pixel (x,y)  {

    cast a ray through that pixel into the scene, and
        find the closest surface along the ray through that pixel

    compute the RGB value, based on that surface
}
```

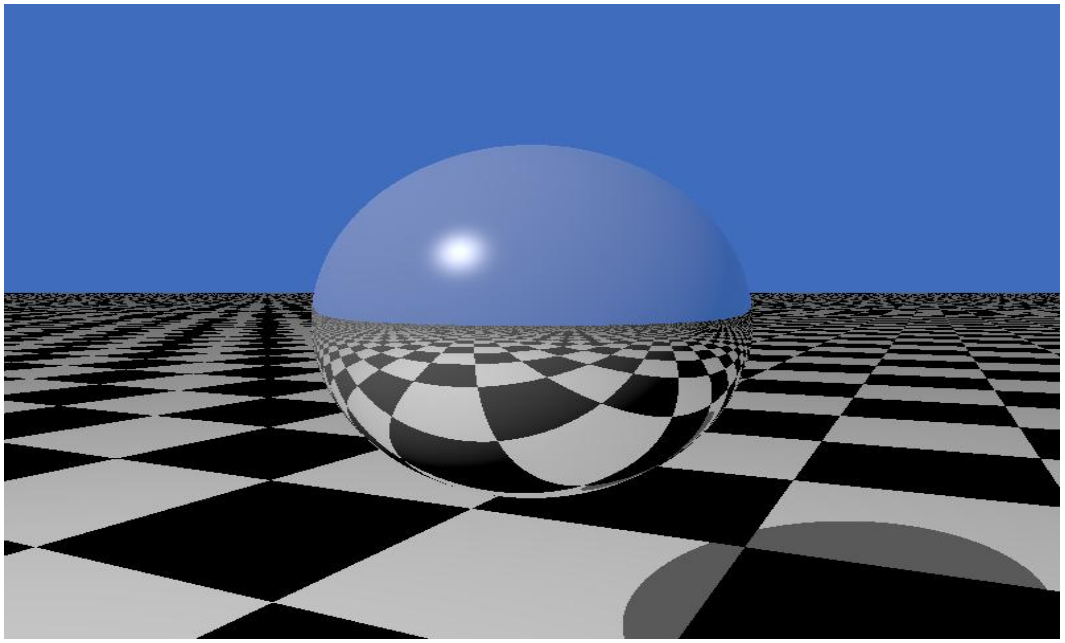
Ray tracing is like ray casting, but now mirror reflections are allowed.

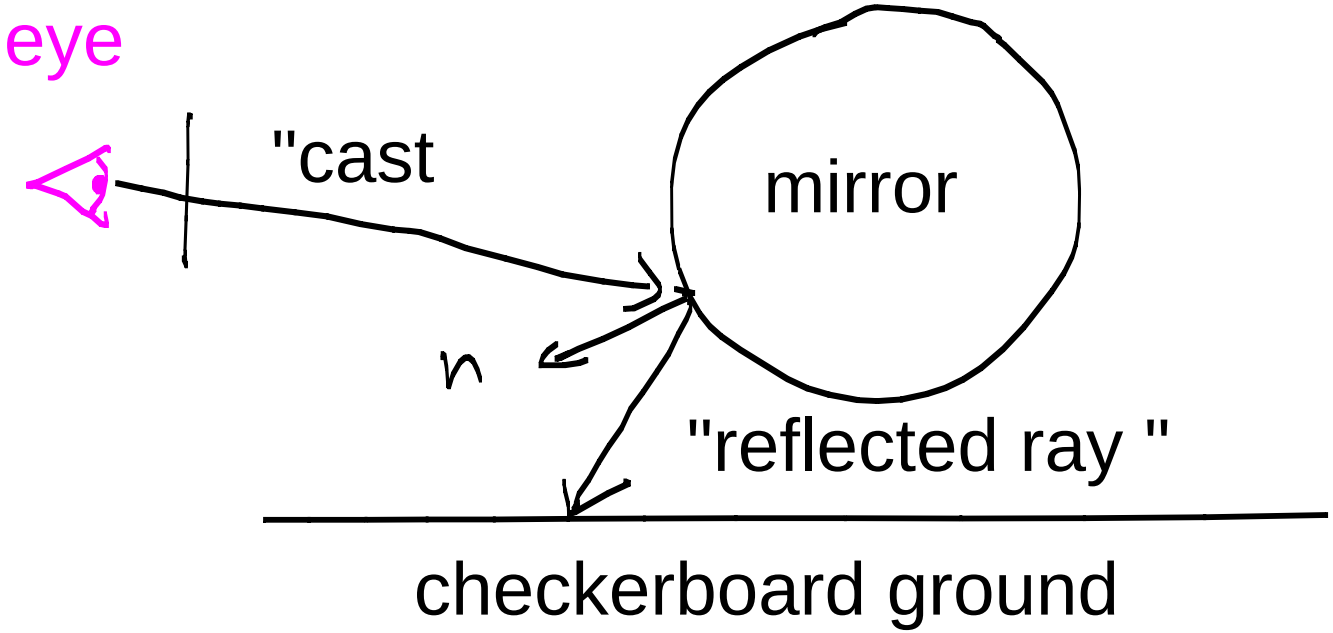


Shadow to be discussed next week.

Ray Tracing (sketch)

```
for each pixel (x,y)  {

    cast a ray through that pixel into the scene

    if  the nearest surface is a mirror,

            repeat
                re-cast the ray in the reflection direction
            until you hit a non-mirror surface

    else
        compute the RGB value of the visible non-mirror
          surface and assign that value to the pixel (x,y)
}

// All the techniques we discussed in lecture 8 can be applied here.
   e.g.   BSP trees,  octrees, hierarchical bounding volumes
```

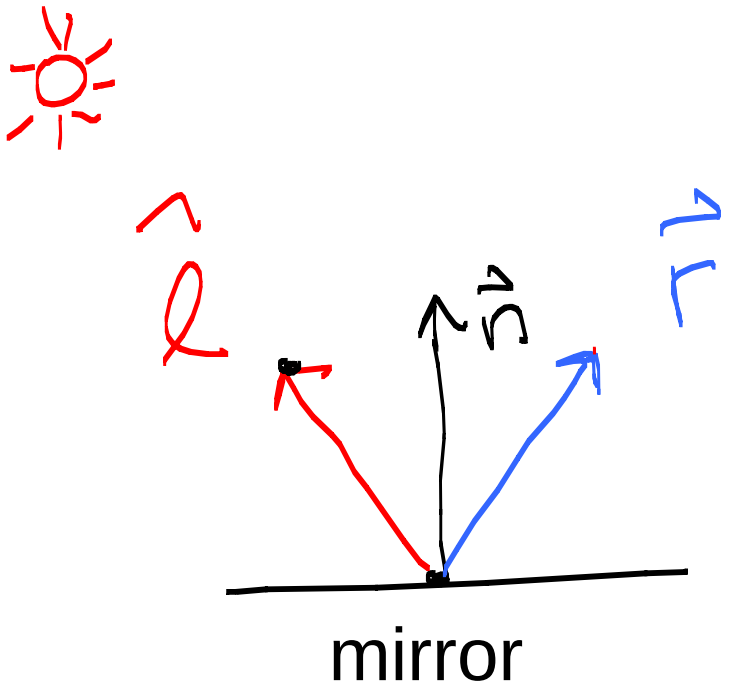



eye
"cast
mirror
n
"reflected ray "
checkerboard ground

Recall lecture 12: Mirror Reflection

$$\vec{r} = 2(\vec{n} \cdot \hat{\ell})\ \vec{n} - \hat{\ell}$$

light

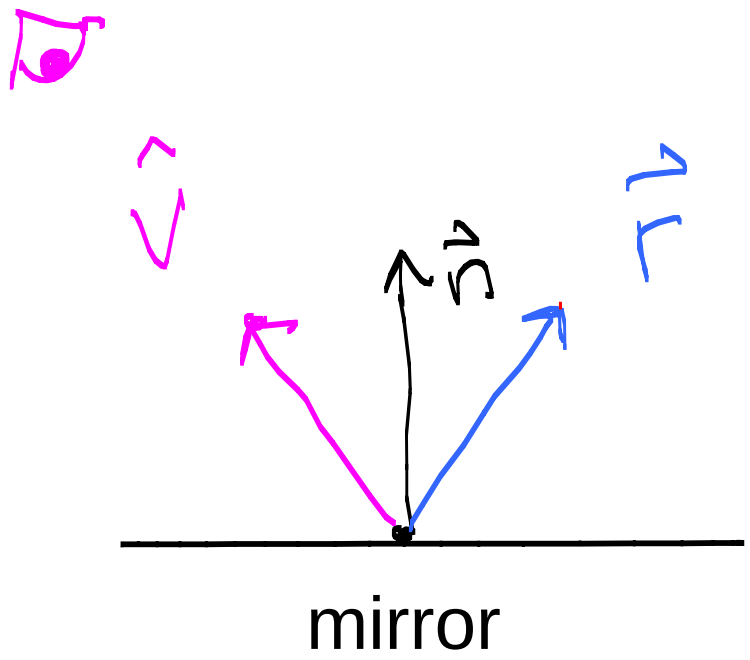


reflection direction

A similar model is used in ray tracing.

$$\vec{r} = 2(\vec{n} \cdot \hat{v})\ \vec{n} - \hat{v}$$

viewer

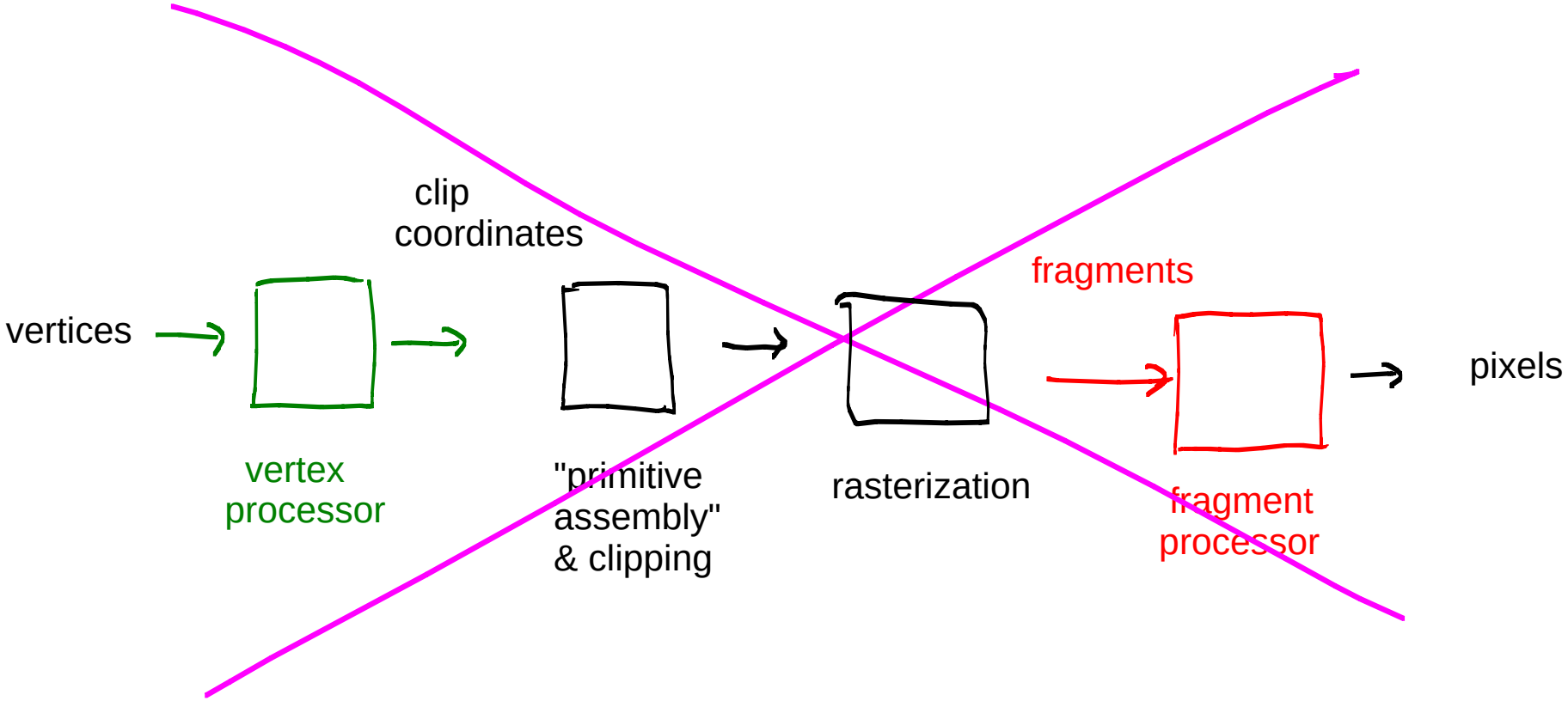


reflection direction

In Assignment 3, you will implement a simple ray tracer.

Part of the material for A3 involves shadows, which we discuss next lecture.

Ray tracing does not naturally fit into graphics pipeline.



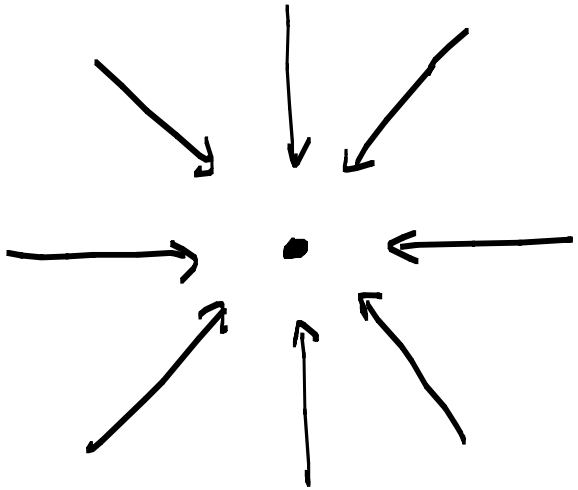
Why not? Because ....vertex and fragment shaders only have access to a small amount of information (in particular, local scene geometry)

Environment Mapping

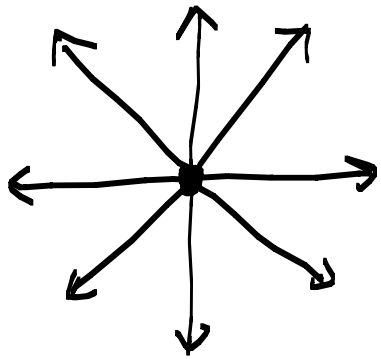
If you want real time graphics with mirror surfaces, use "environment mapping" instead.

An "environment map" is a "360 deg" RGB image of a scene, defined on the unit sphere centered at some scene point.

We denote it $E(r_x, r_y, r_z)$ where **r** is direction of rays to the environment as seen from the scene point.



physical situation



$E(r_x, r_y, r_z)$

Environment Mapping Algorithm

```
for each pixel (xp,yp) {

   cast a ray through (xp, yp) to the nearest scene point (x,y,z)

   if (x,y,z) has mirror reflectance {
         compute mirror reflection direction (rx, ry, rz)

         //  copy environment map value
         I(xp, yp) = E(rx, ry, rz)
    }
    else
          compute RGB using Blinn-Phong (or other model)
}
```

How does this differ from ray tracing ?

Example (Blinn and Newell 1976)

hand drawn (cylindrical) environment by Jim Blinr

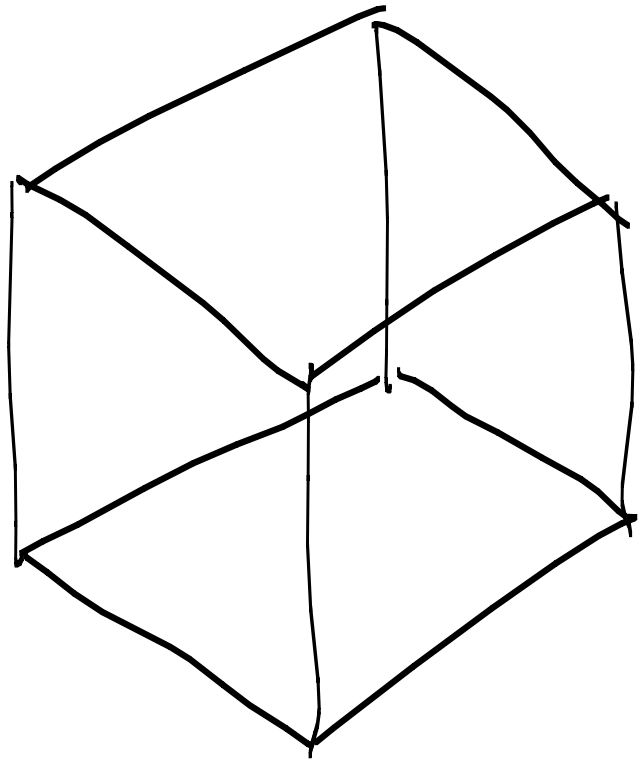
Notice reflections of windows in the teapot.

This teapot has diffuse shading + mirror components.
Today we will talk only about the mirror component.

A more complex environment...

Cube map (Green 1986)

a data structure for environment maps



$x = \pm 1$

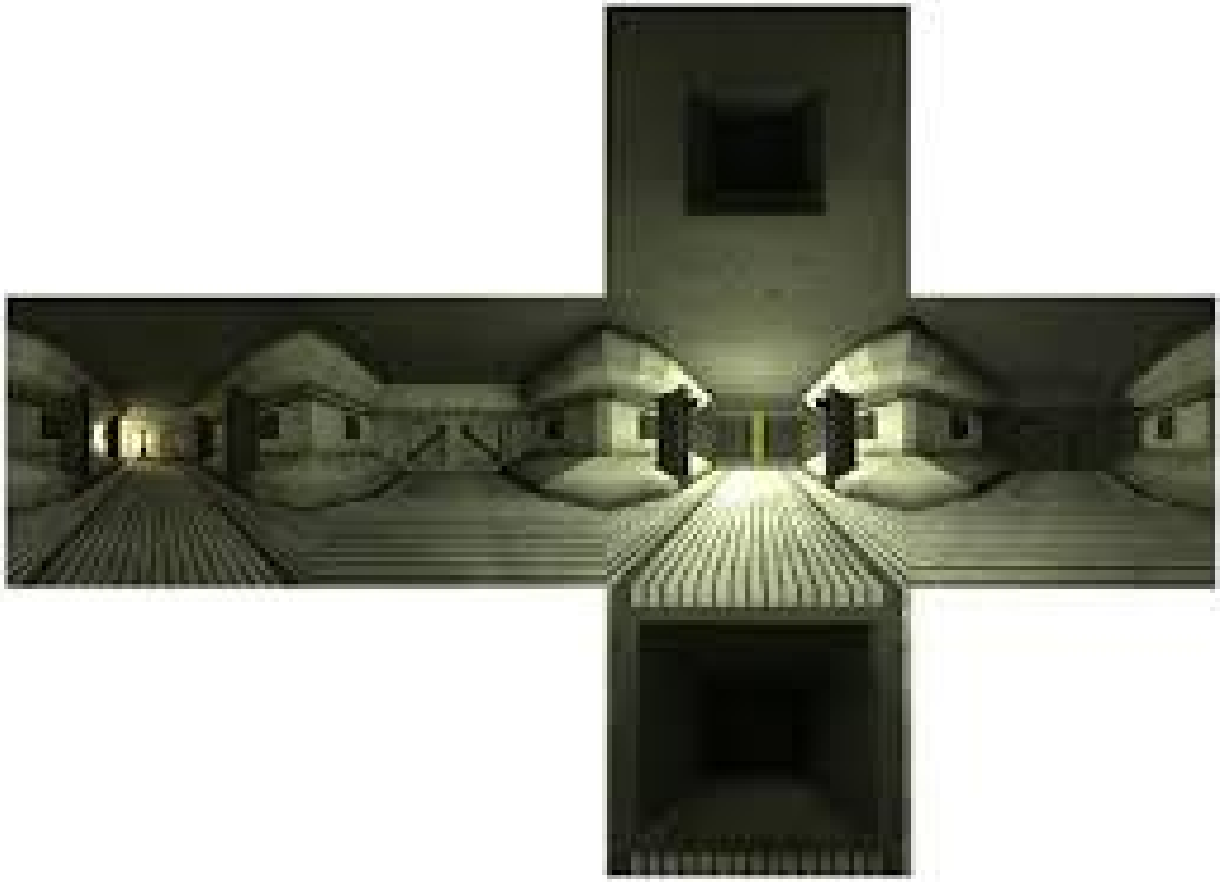
$y = \pm 1$

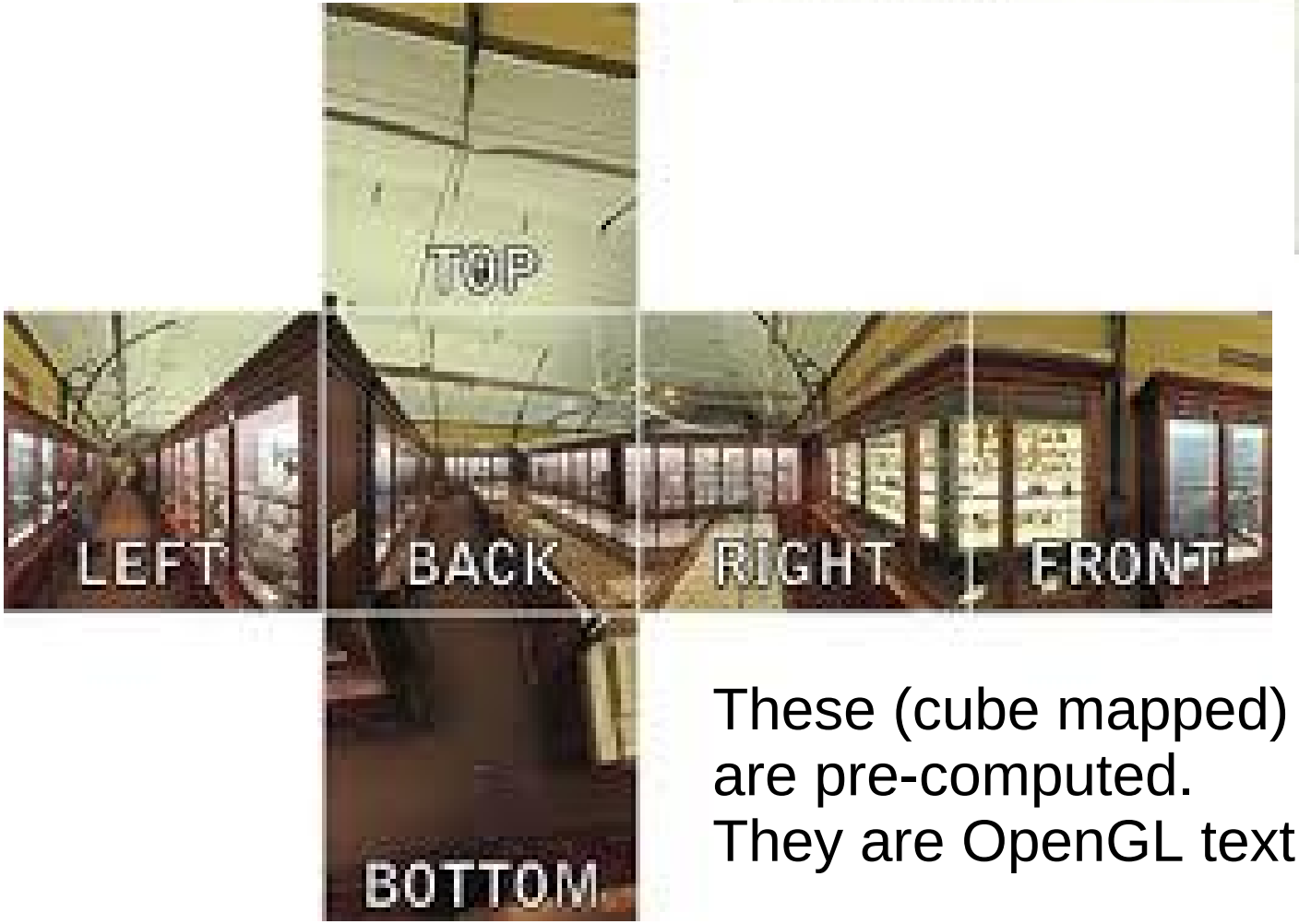
$z = \pm 1$

Construct 6 images, each with 90 deg field of view angle.

Examples

google "cube map images"



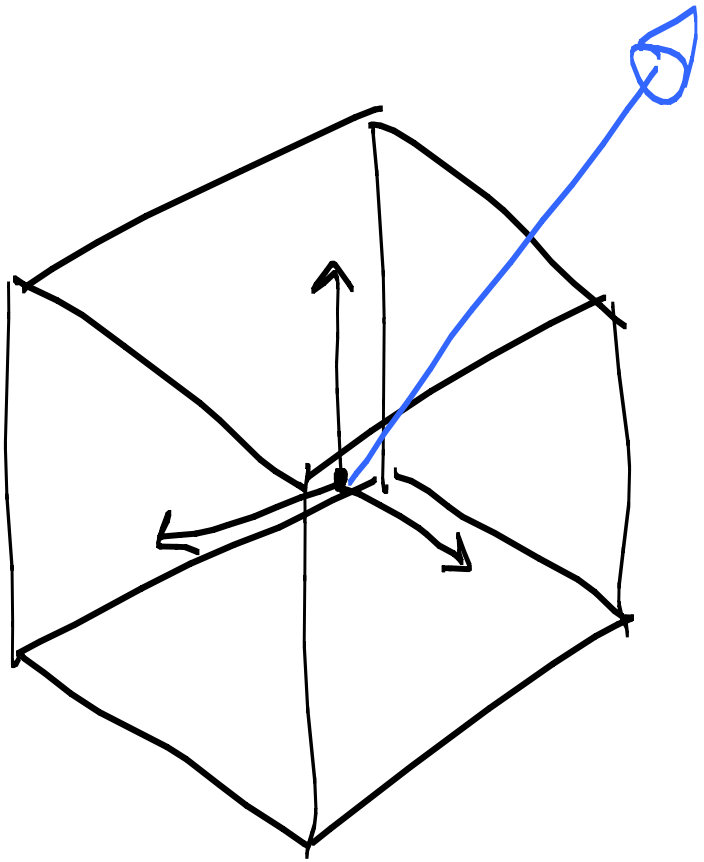


These (cube mapped) environment maps are pre-computed.
They are OpenGL textures.

How to index into a cube map? (derivation over next few slides)

Given reflection vector **r** = (r_x, r_y, r_z), lookup E(r_x, r_y, r_z)

How ?



$x = \pm 1$

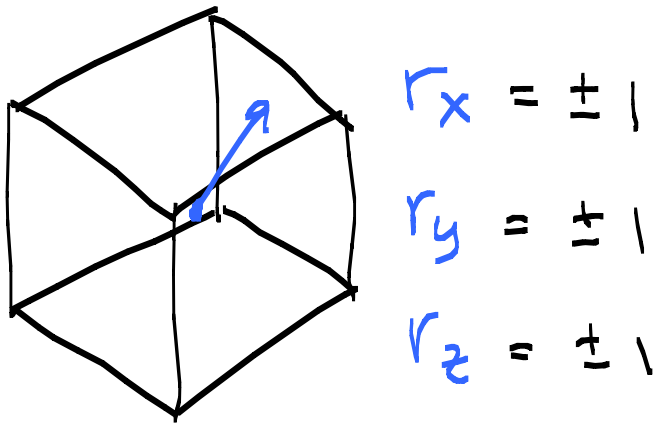
$y = \pm 1$

$z = \pm 1$

A: Project **r** onto cube face.

$$(r_x, r_y, r_z) = (r_x, r_y, r_z) / \max \{ | r_x | , | r_y | , | r_z | \}.$$

One of the components will have value +- 1, and this tells us which face of the cube to use. The other two components are used for indexing into that face.



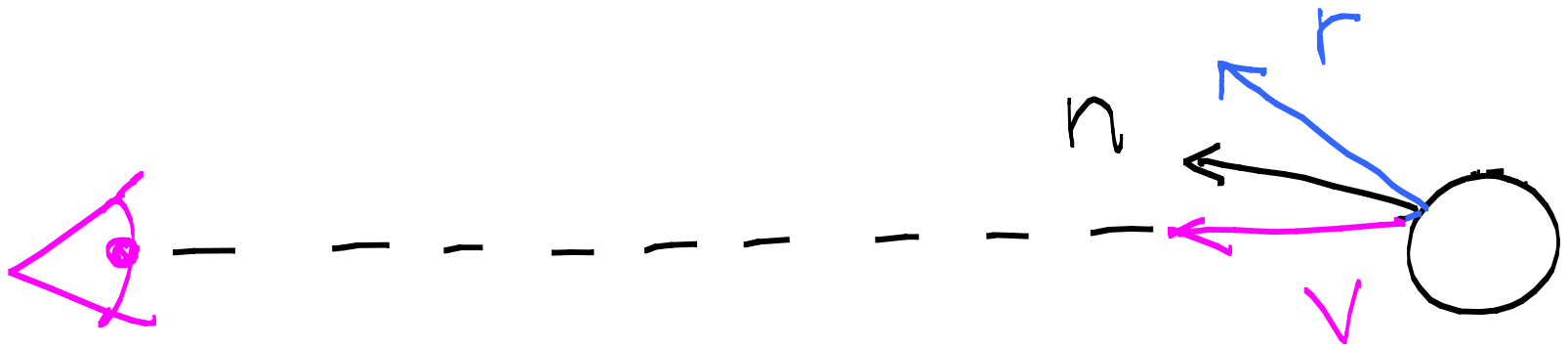
Need to remap the indices from [-1, 1] to [0, N) where each face of the cube map has NxN pixels.

Sphere Map

a second data structure for environment maps

The reflection of the scene in a small mirror sphere.

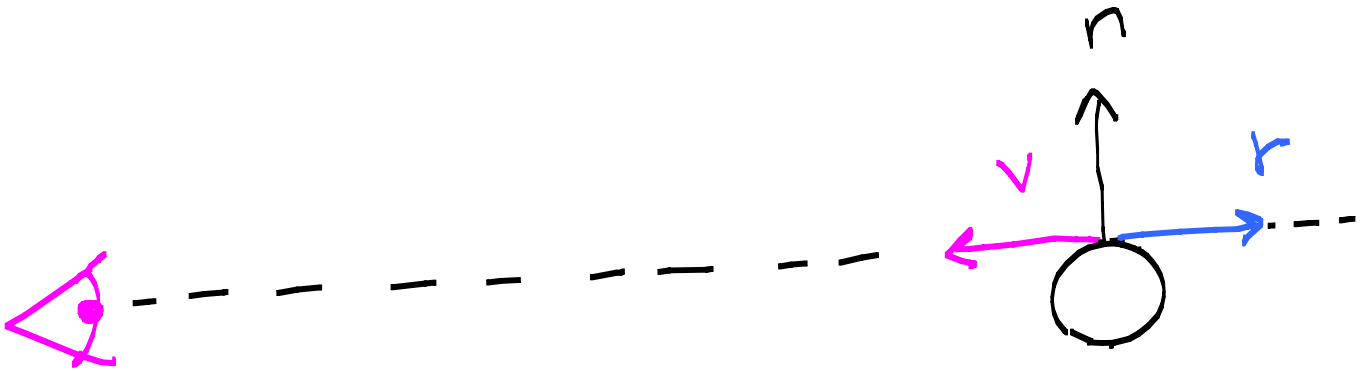
Q: How much of scene is visible in the reflection off a mirror sphere ?

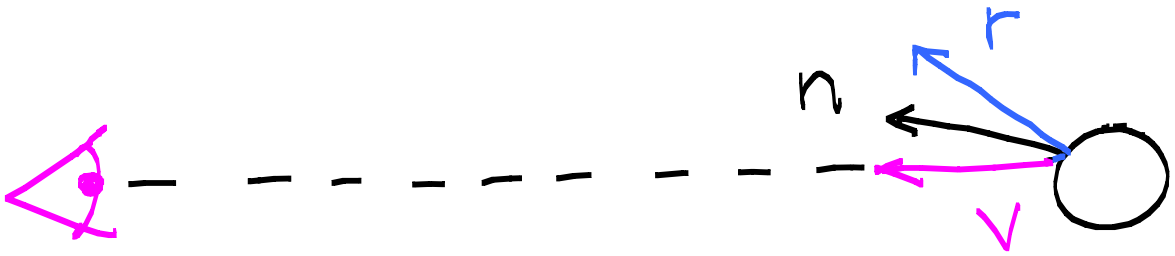


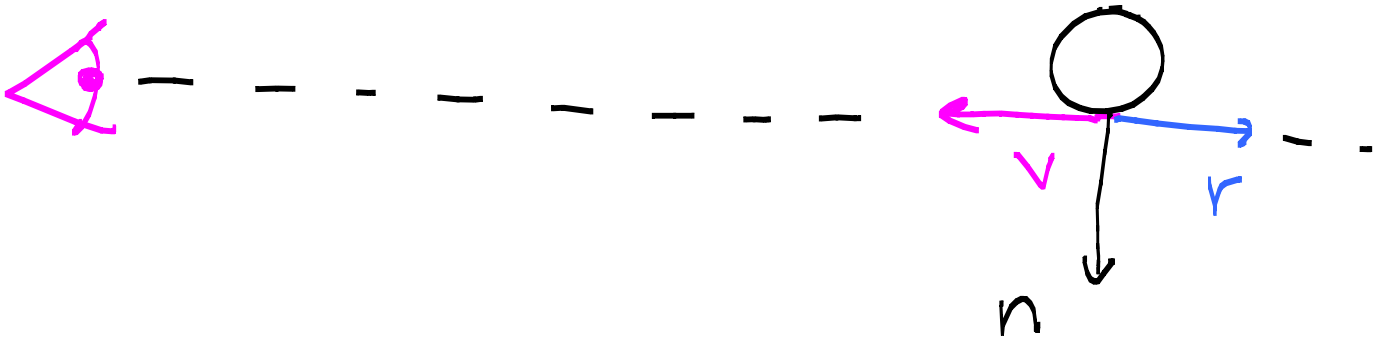
A: - 180 degrees ? (half of the scene)

- between 180 and 360 degrees ?

- 360 degrees ? (the entire scene)

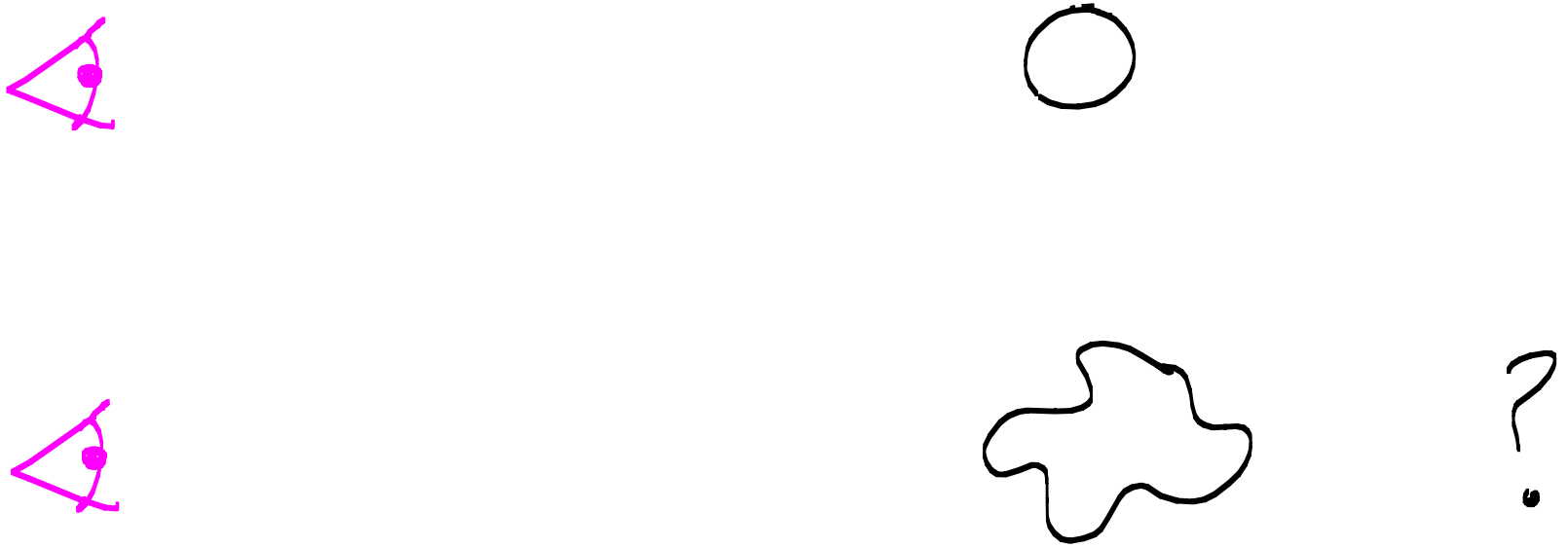

n
v
r


r
n
v


v
r
n

Claim: if you are given the reflection of a scene in a small spherical mirror, then you can compute what any mirror object looks like (if know the object's shape).

Why? All you need to know is the reflection direction at each pixel and then use it to index index into the environment map defined by the given image of the sphere.

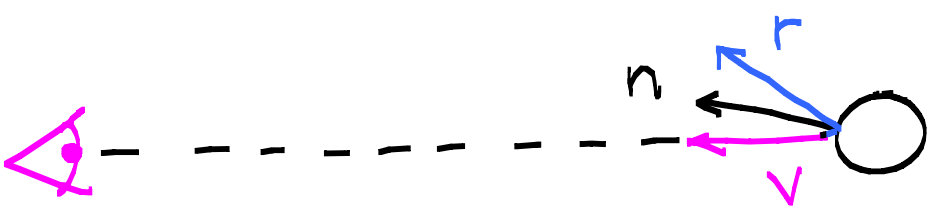


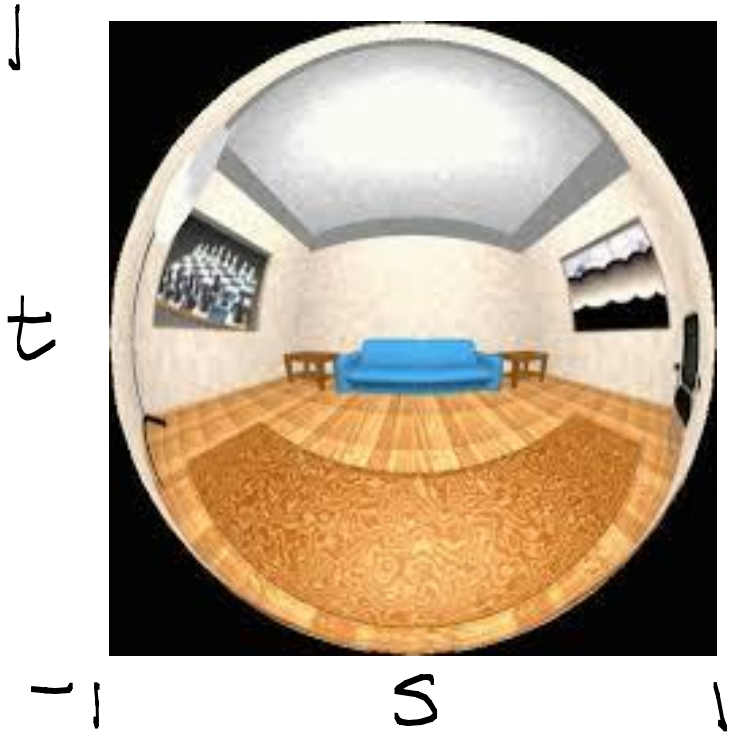
How to index into a sphere map?

Given reflection vector r = (rx, ry, rz), lookup E(rx, ry, rz)

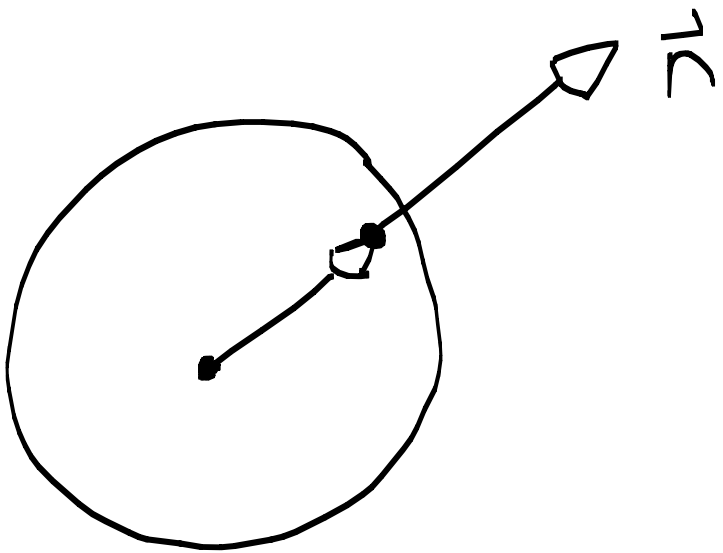
Q: How ?

A: Map r = (rx, ry, rz) to (s, t) which is defined below.





Let's now derive the mapping. First observe:

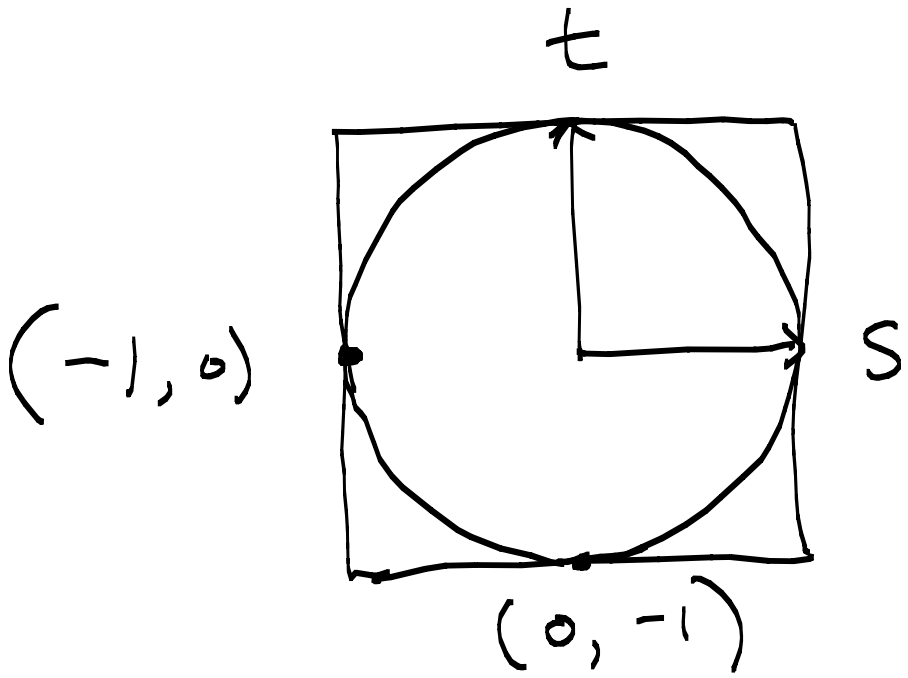


For a point on a unit sphere,

$$(x, y, z) = (n_x, n_y, n_z)$$

$$= \left(x, y, \sqrt{1 - x^2 - y^2}\right)$$

Texture coordinates for a sphere map



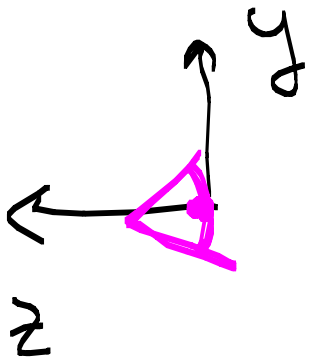
Identify (s,t) coordinates with (n_x, n_y).

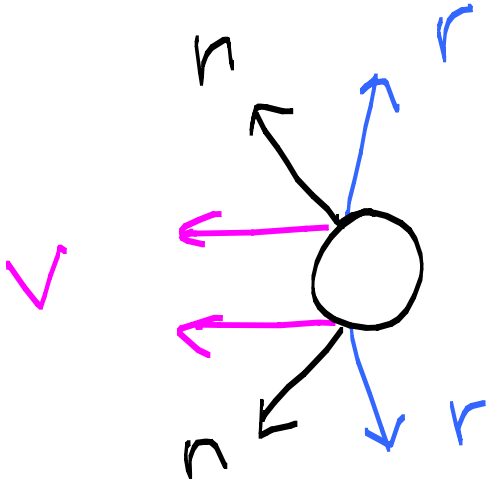
$$\vec{n} = (n_x, n_y, n_z) = (s, t, \sqrt{1-s^2-t^2})$$

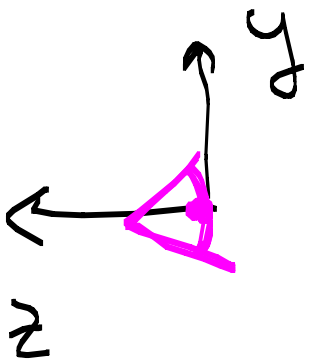
To map $\mathbf{r} = (r_x, r_y, r_z)$ to (s,t),

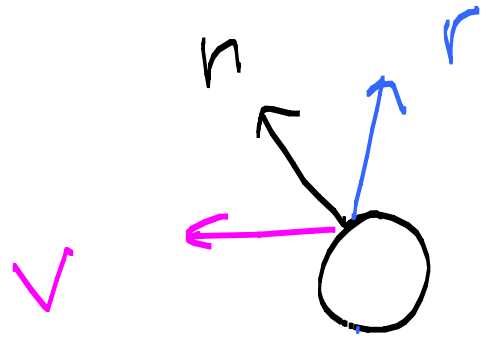
assume orthographic projection,

namely assume $\mathbf{v} = (0, 0, 1)$ for all surface points.









$$\vec{r} = 2(\vec{n} \cdot \vec{v})\vec{n} - \vec{v}$$

Substituting for **v** and **n** and manipulating gives:
(details in lecture notes)

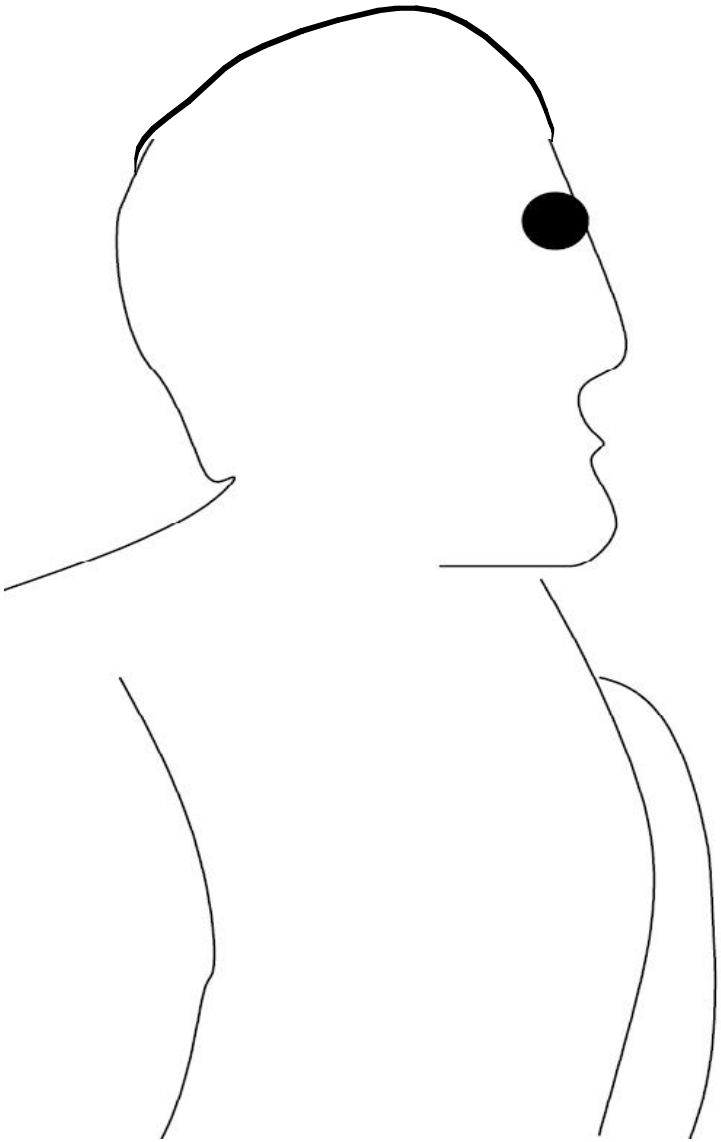
$$(s, t) = \frac{1}{\sqrt{r_x^2 + r_y^2 + (r_z+1)^2}} (r_x, r_y)$$

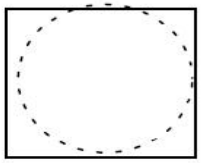
Capturing Environment Maps with Photography

early 1980s http://www.pauldebevec.com/ReflectionMapping/

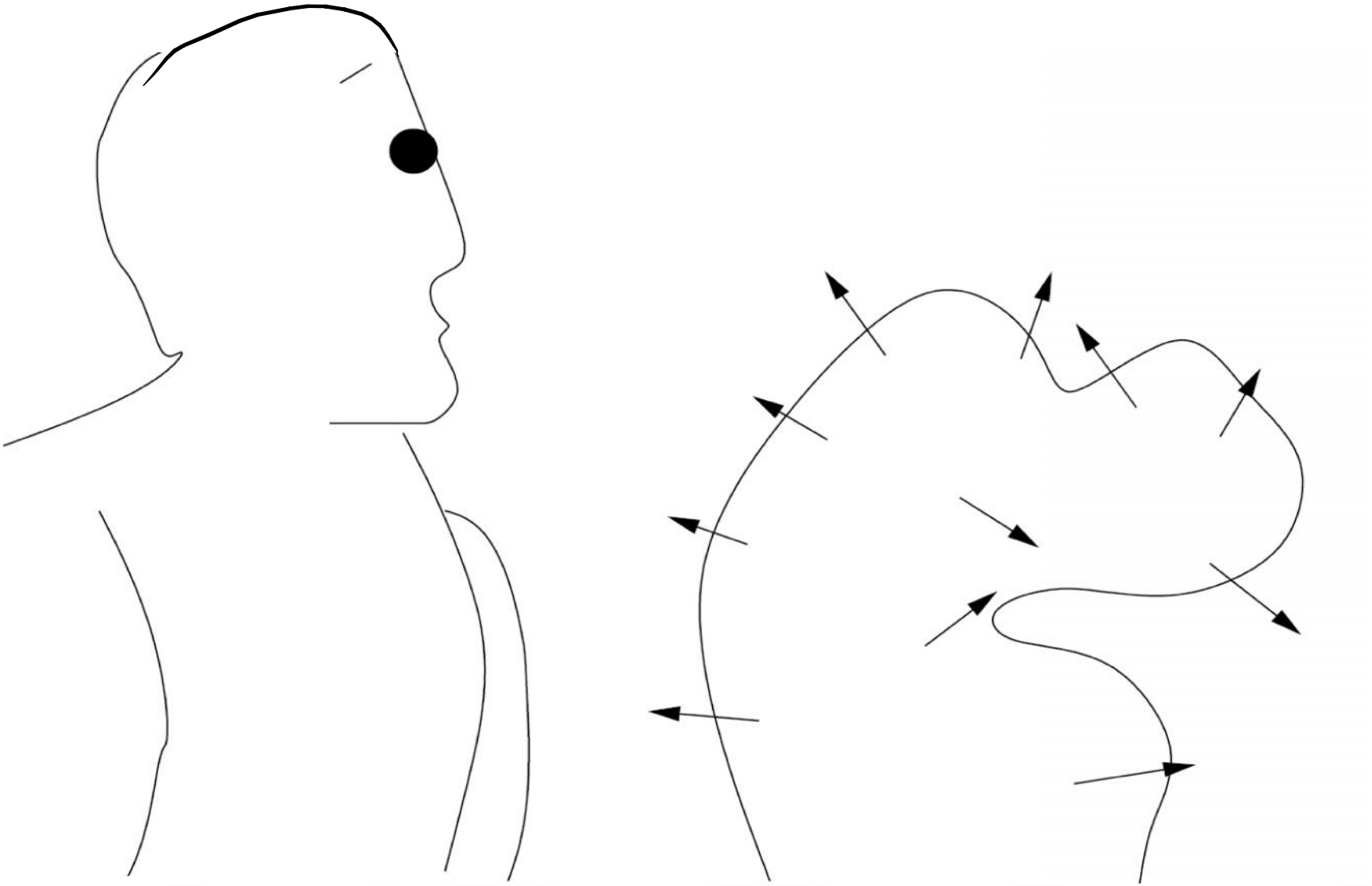
Hollywood: Terminator 2 (1991)

How was this achieved?





mirror sphere



Q: What implicit assumptions are we making when using environment maps ?

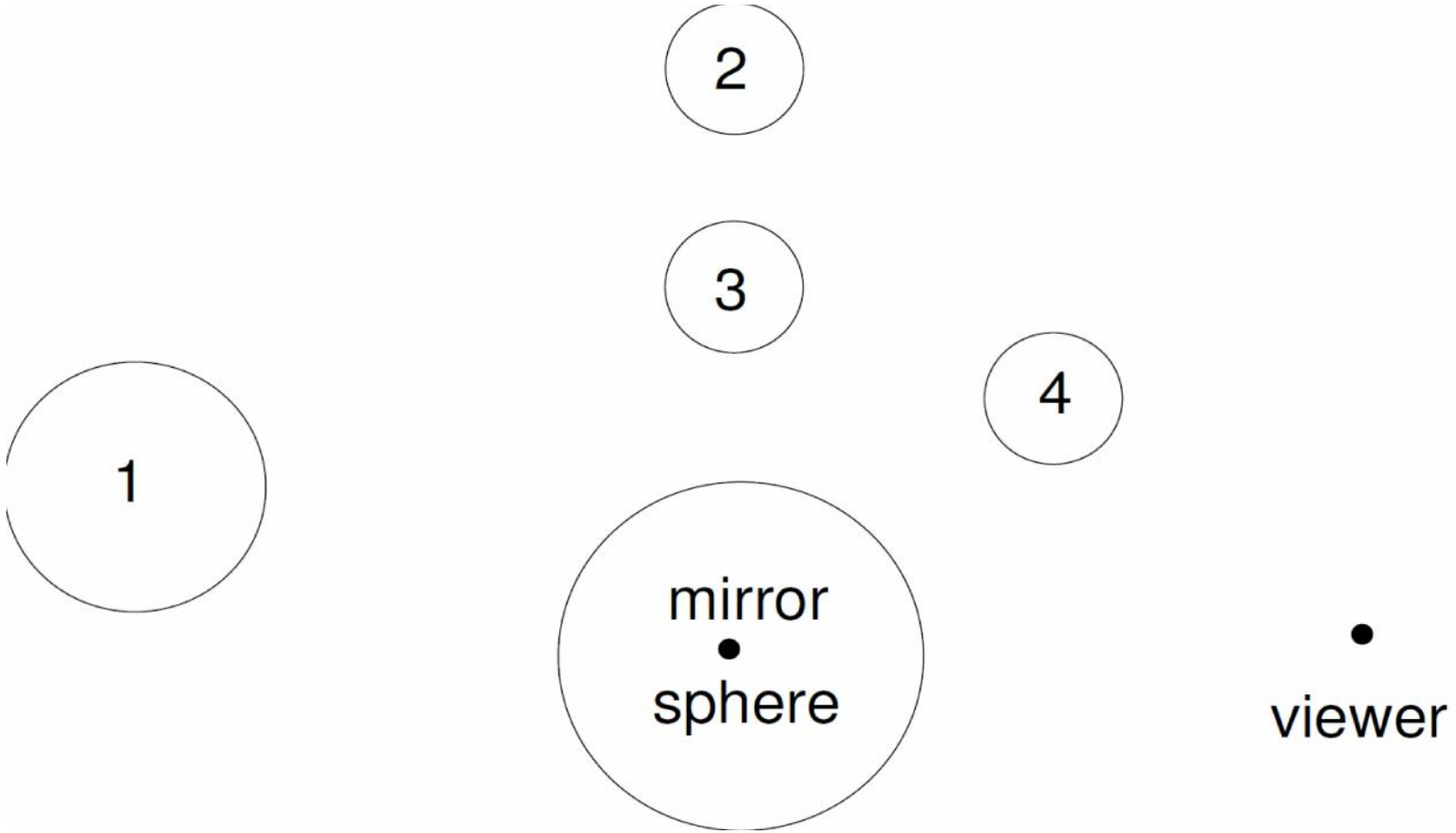
A: We assume the environment is the same for all scene points (x,y,z). This essentially assumes that the environment is at infinity.

Why is that wrong?

Why does it not matter ?

Q: For the scene below, which of the objects (1, 2, 3, 4) would be visible or partly visible either directly or in the mirror, if we used:

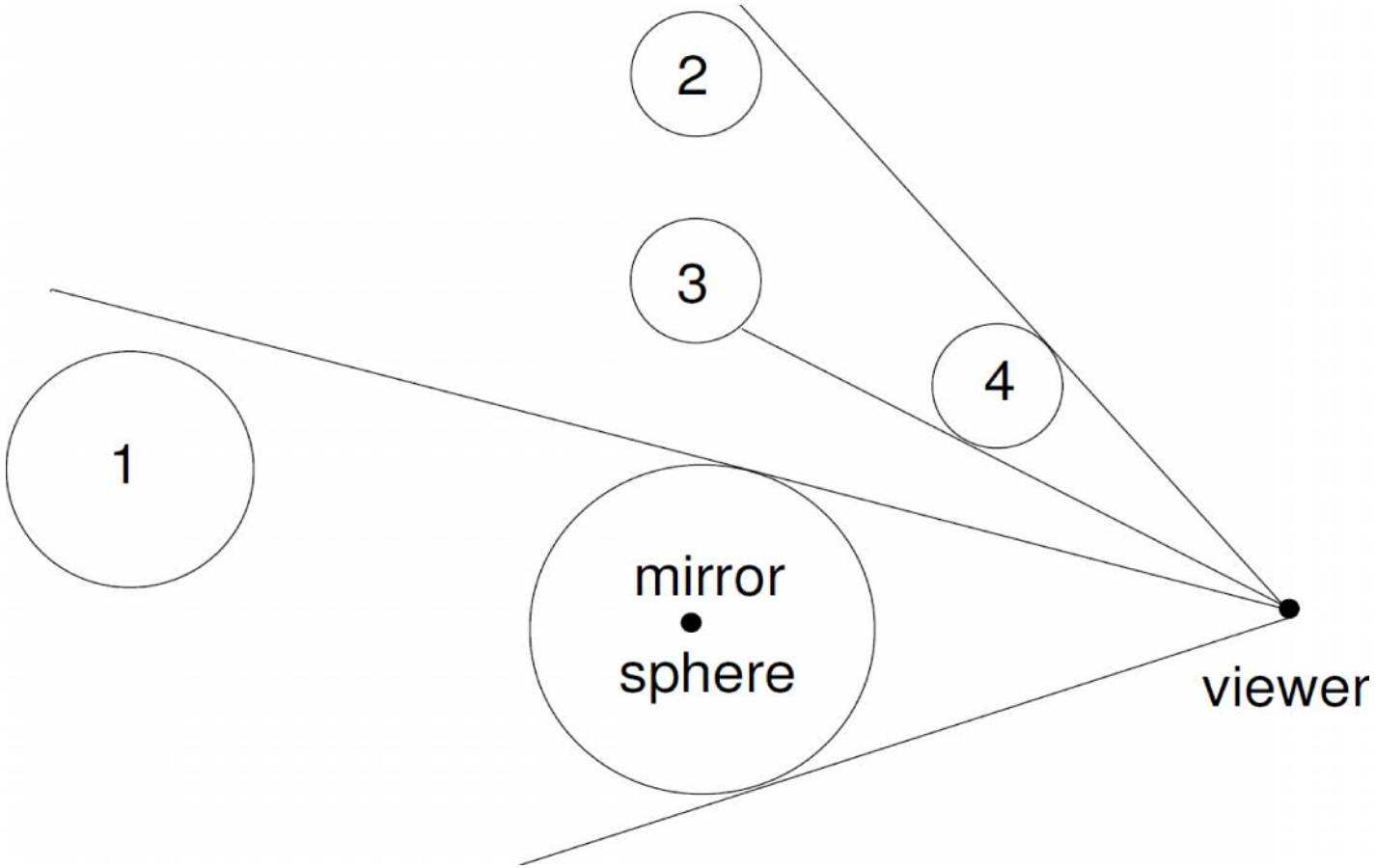
- ray tracing
- environment mapping



Q: For the scene below, which of the objects (1, 2, 3, 4) would be visible or partly visible directly, if we used:

- ray tracing
- environment mapping

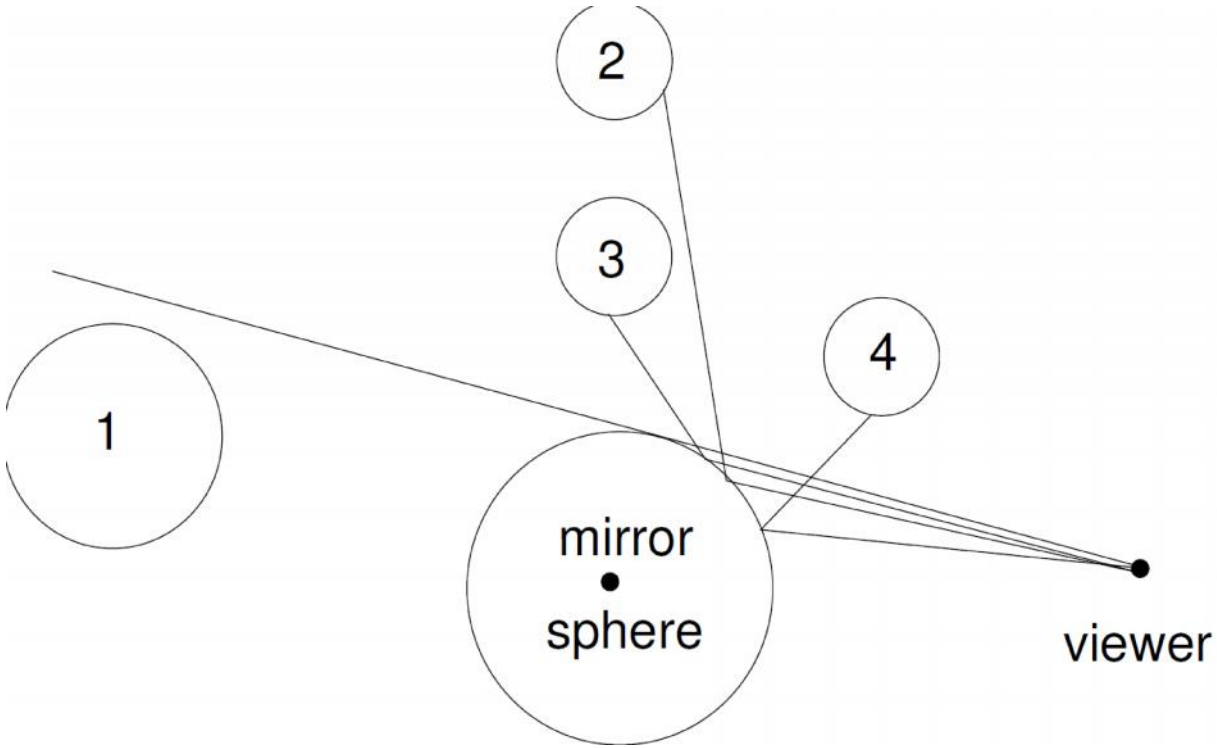
A: 3, 4 (but not 1, 2)



Q: For the scene below, which of the objects (1, 2, 3, 4) would be visible or partly visible in the mirror, if we used:

- ray tracing
- environment mapping

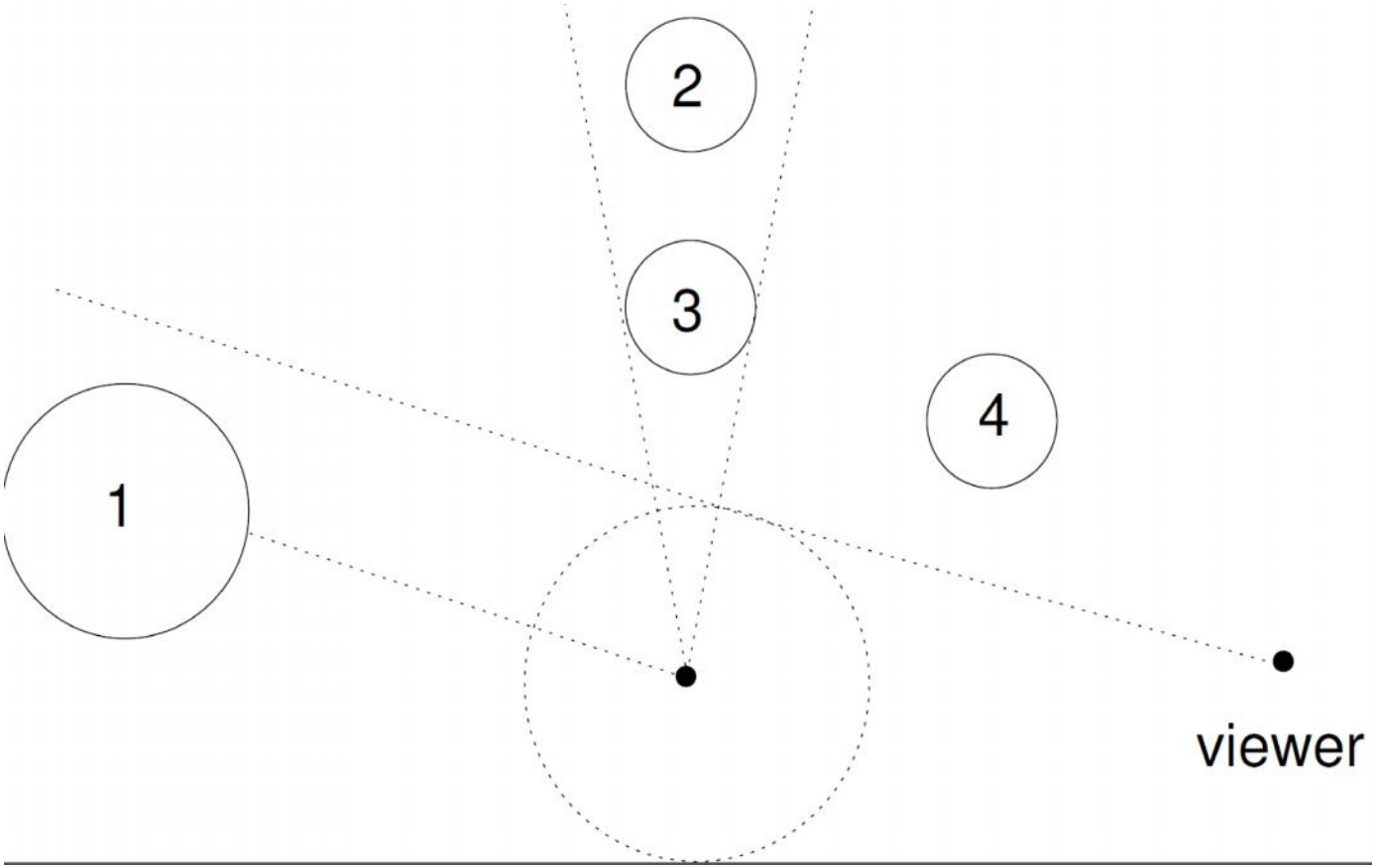
A: 2, 3, 4 (but not 1)

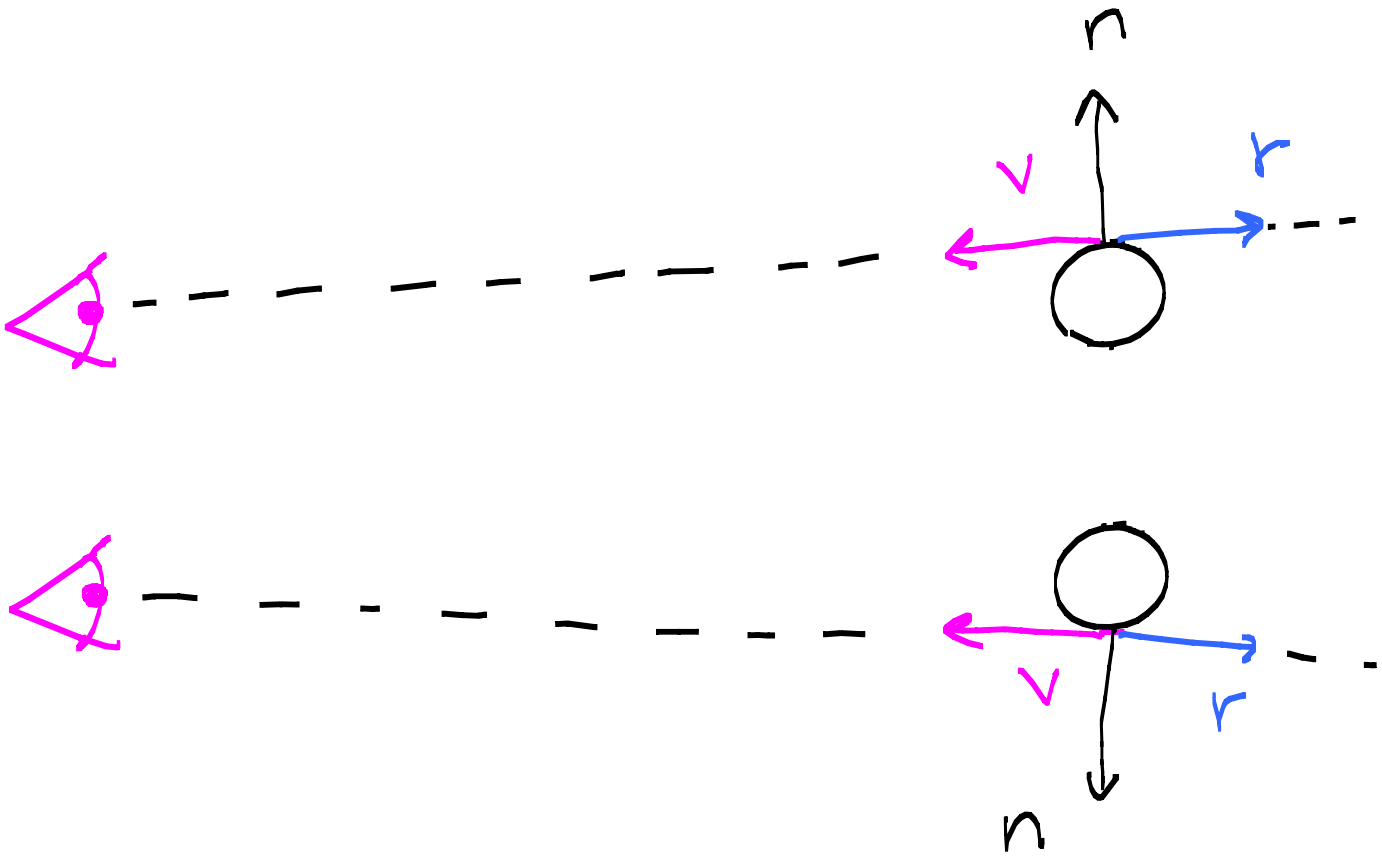


Q: For the scene below, which of the objects (1, 2, 3, 4) would be visible or partly visible in the mirror, if we used:

- ray tracing
- environment mapping

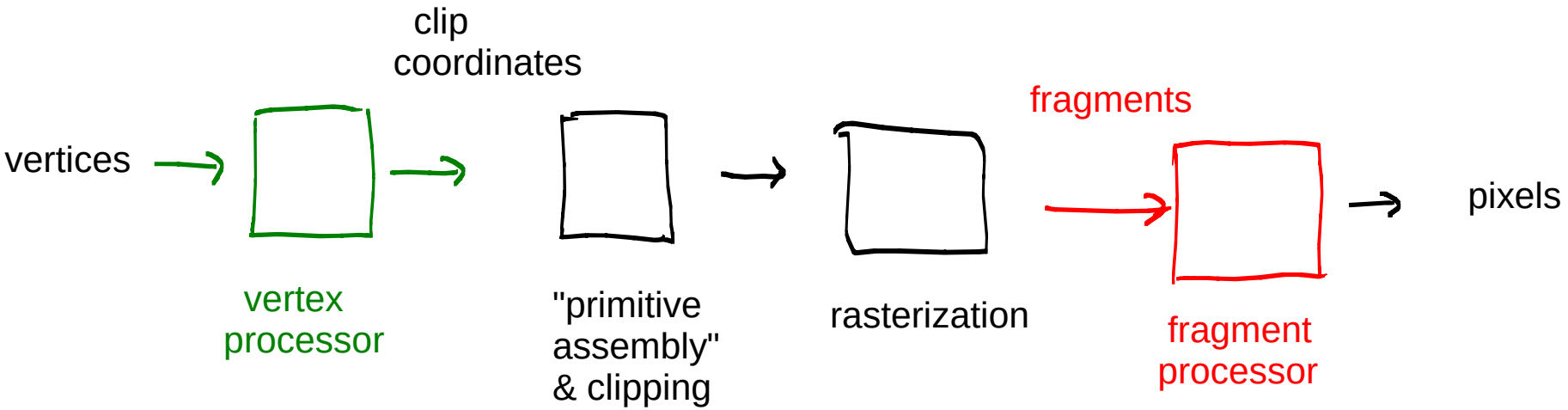
A: 1, 3, 4 (but not 2)

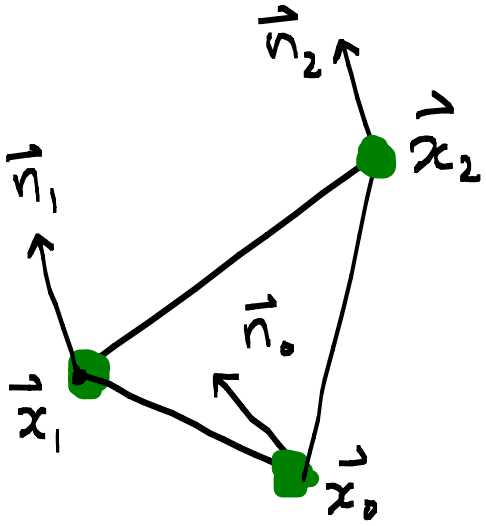


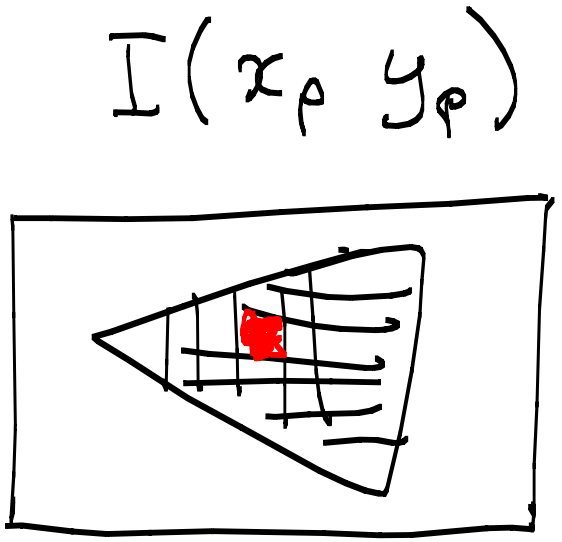


Notice the poor resolution at the rim of the circle.

Where do various steps of *environment mapping* occur in the graphics pipeline?



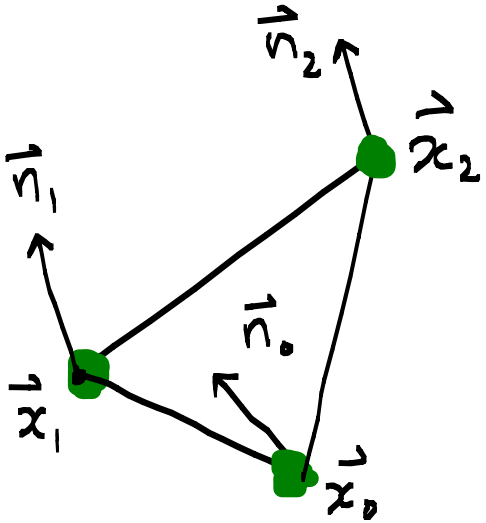


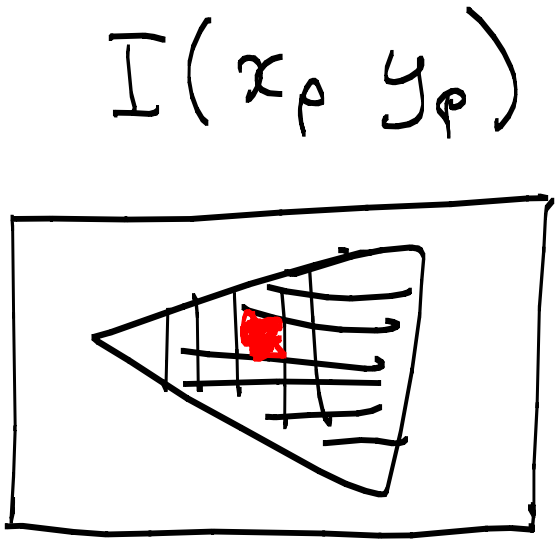


Vertex processor ... ?

Rasterizer .... ?

Fragment processor ... ?





Solution 1 (bad, not used):

Vertex processor computes the reflection vector **r** for each vertex, and looks up RGB values from environment map.

Rasterizer interpolates RGB values and assigns them to the fragments.

Fragment processor does basically nothing.

Why is this bad ?
(Hint: think of what would happen for a square mirror. It would smoothly interpolate between the values at the four corners. That's not what real mirrors do.)

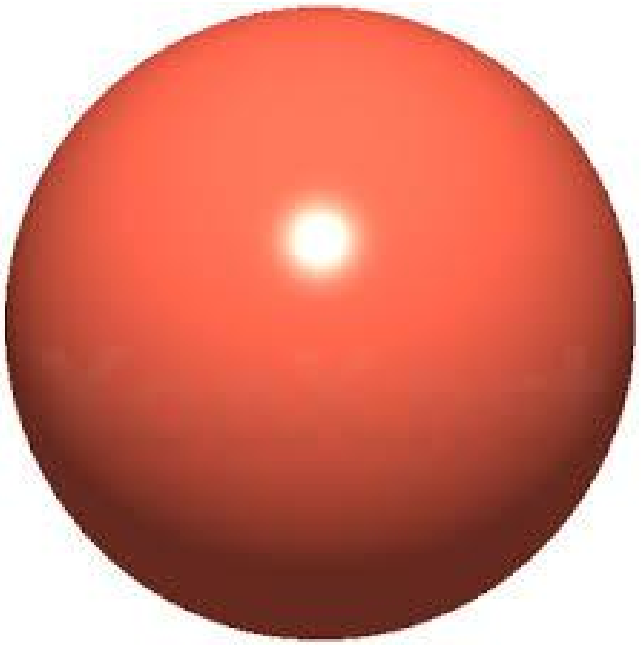
Solution 2 (good):

Vertex processor computes the reflection vector **r** for each vertex.

Rasterizer interpolates reflection vectors and assigns them to the fragments.

Fragment processor uses reflection vectors to look up RGB values in the environment map.

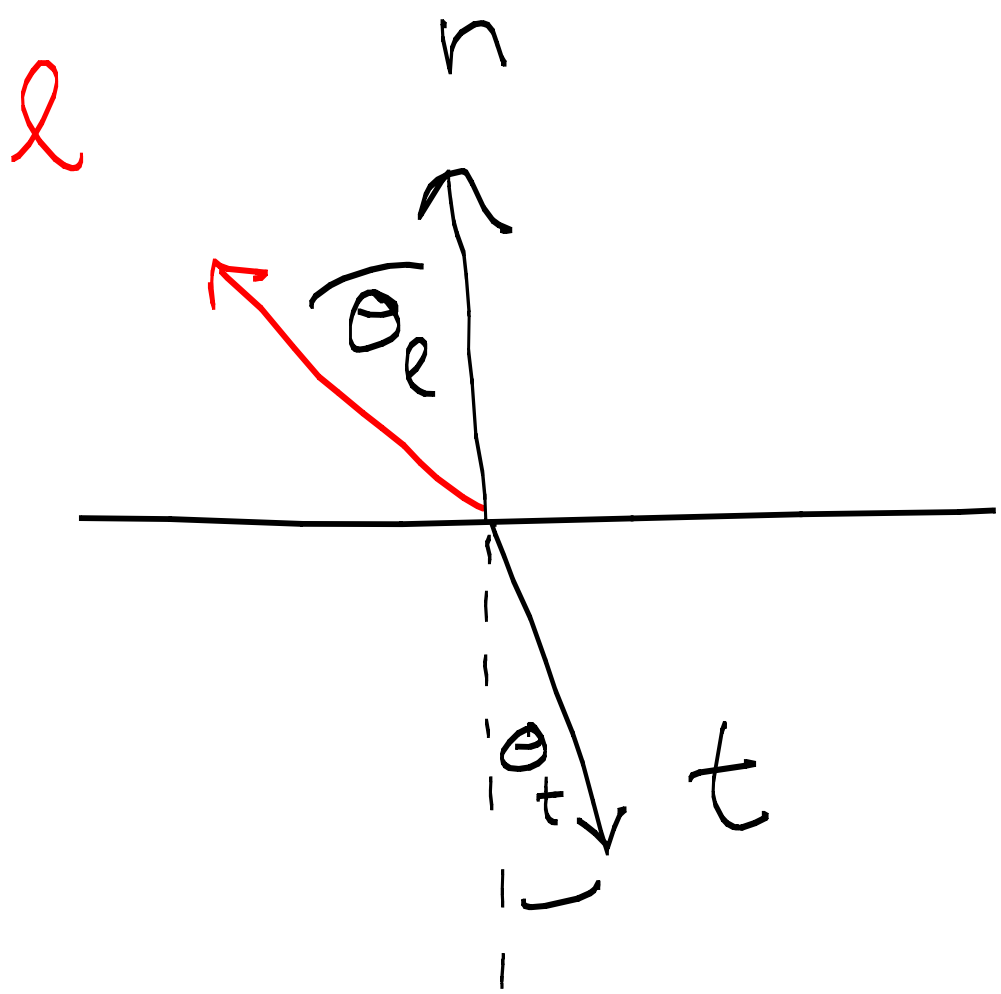
Any image of a sphere can be used as an environment map.



lecture 18

- ray tracing
- environment mapping
- refraction

Refraction



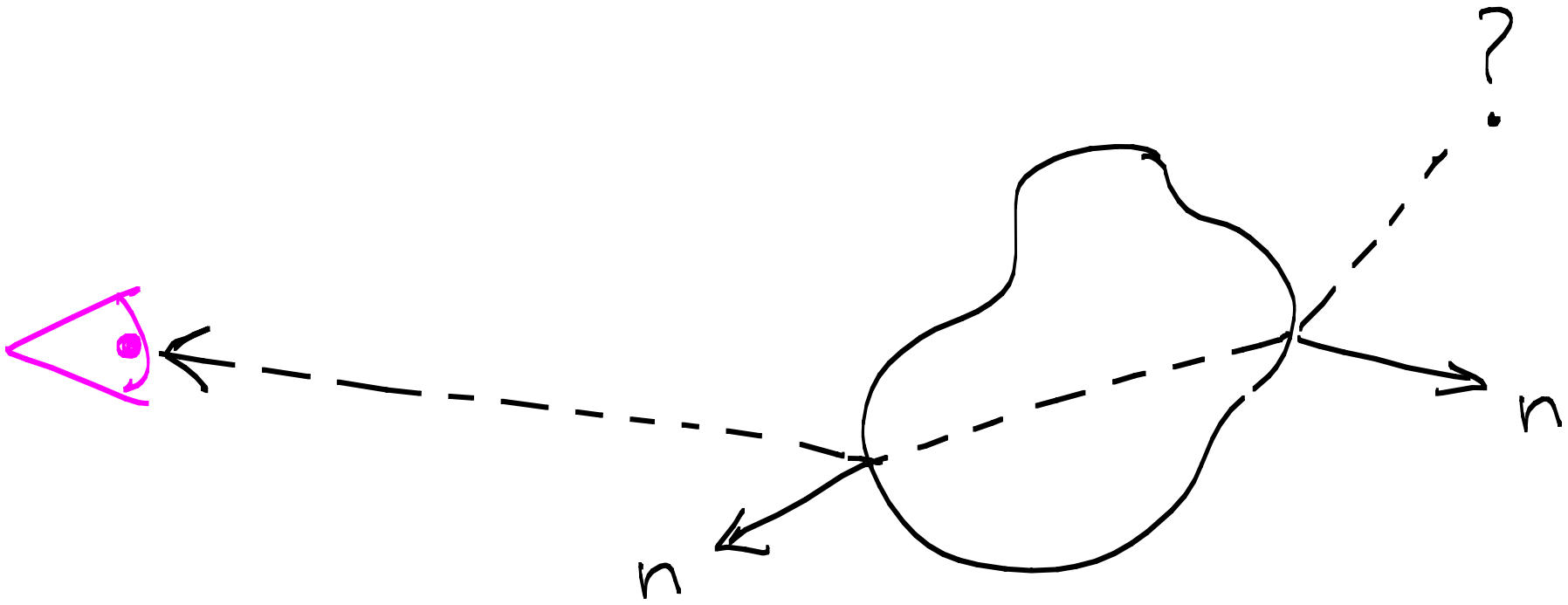
k_i

k_t

index of refraction

Snell's Law

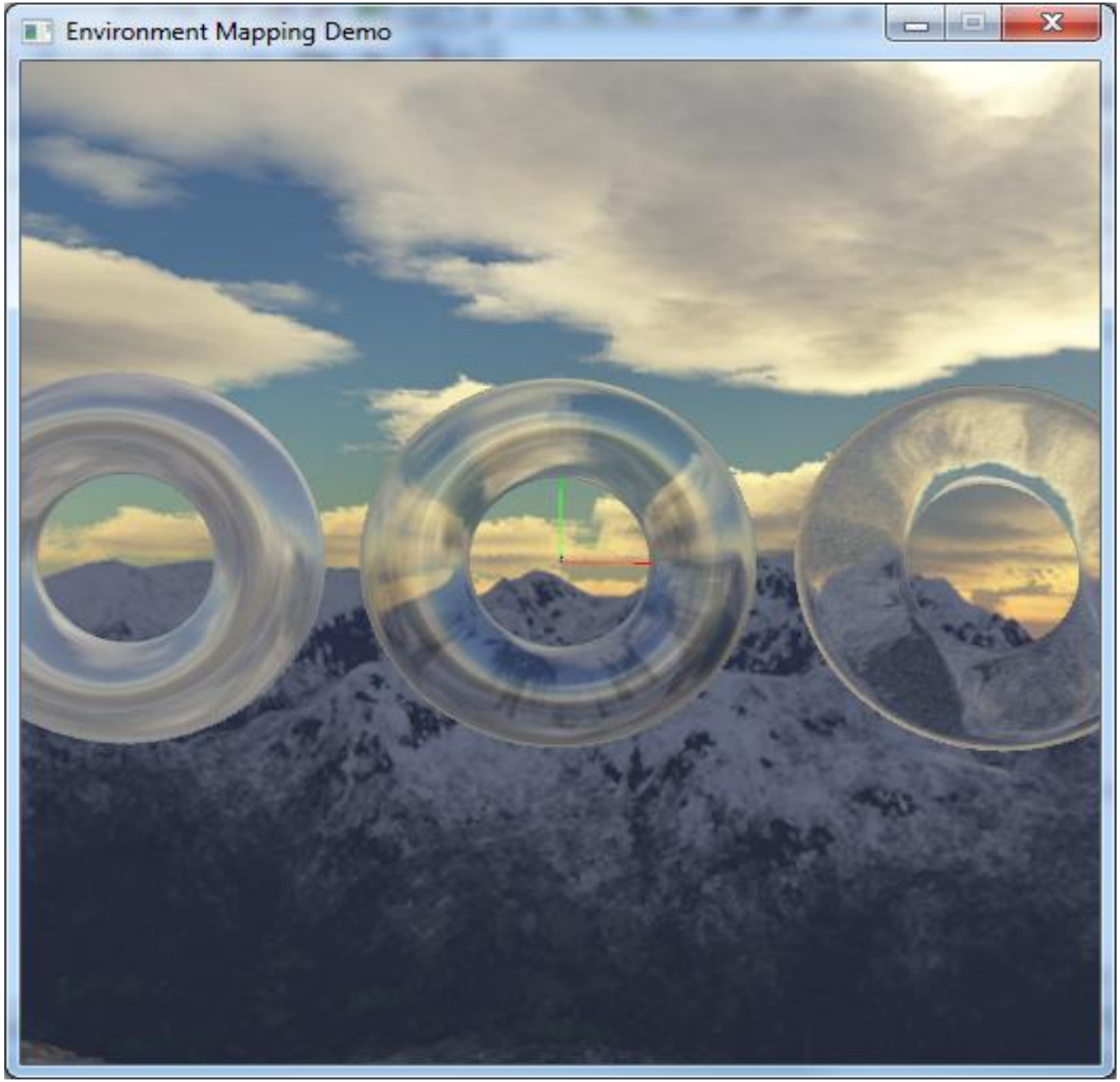
$$\frac{k_i \sin\theta_i}{k_t \sin\theta_t} = 1$$



You can use environment mapping or ray tracing to get the RGB value.

Environment mapping can be done in OpenGL 2.x and beyond using shaders.

.



See nice video here.
http://3dgep.com/environment-mapping-with-cg-and-opengl/#The_Refraction_Shader

Geri's Game (1997, Pixar, won Oscar for animation)

https://www.youtube.com/watch?v=9IYRC7g2ICg

Check out refraction in glasses around 2 minutes in.
This was done using environment mapping.