Computer Graphics

Lecture 9
Environment mapping, Mirroring

Today

Environment Mapping

- Introduction
- Cubic mapping
- Sphere mapping
- refractive mapping

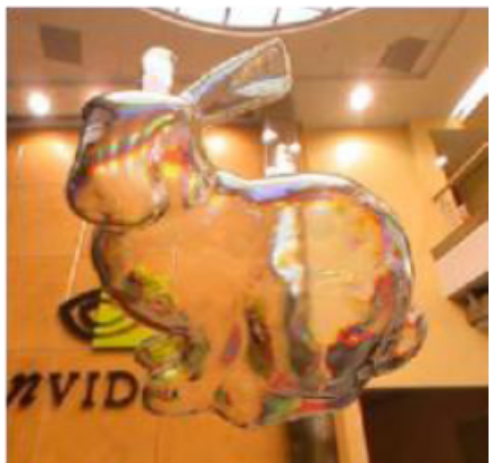
Mirroring

- Introduction
- reflection first
- stencil buffer
- reflection last

Environment Mapping : Background

Many objects in the world are glossy or transparent

- Glossy objects reflect the external world
- The world is refracted through the transparent objects
- Important to make the virtual scene to appear realistic



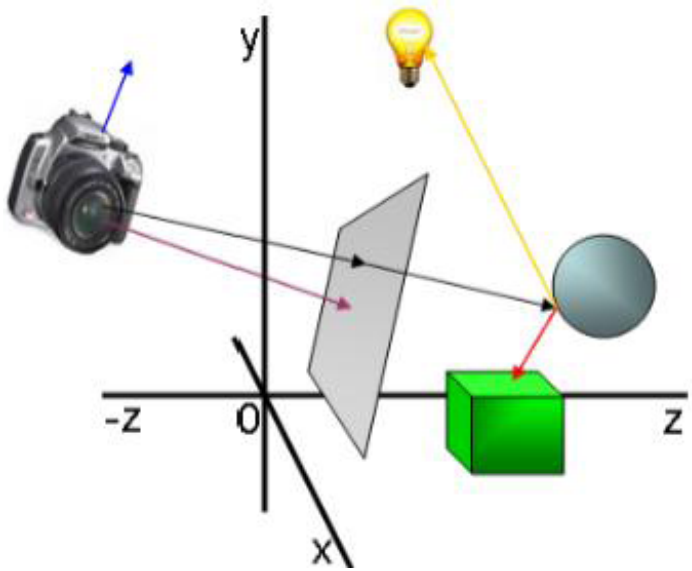
Example

Terminator II

Environment Mapping: Background (2)

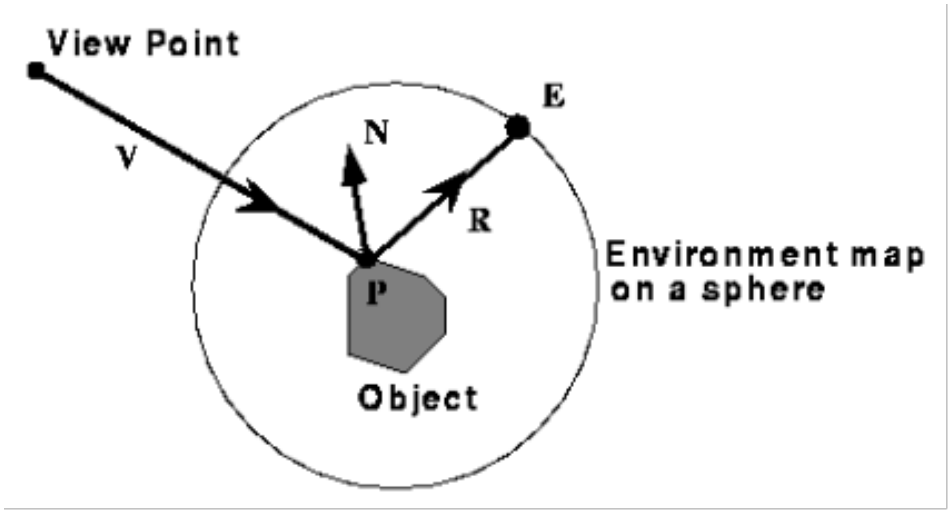
Precisely simulating such phenomena is computationally costly

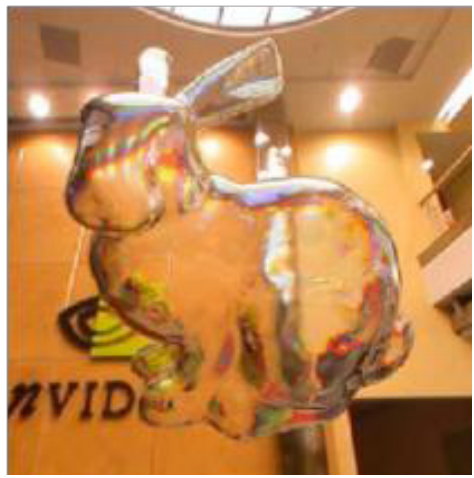
- Requires ray tracing, which can be expensive
- Tracking the rays and finding out where they collide, further doing a lighting computation there



Environment Mapping

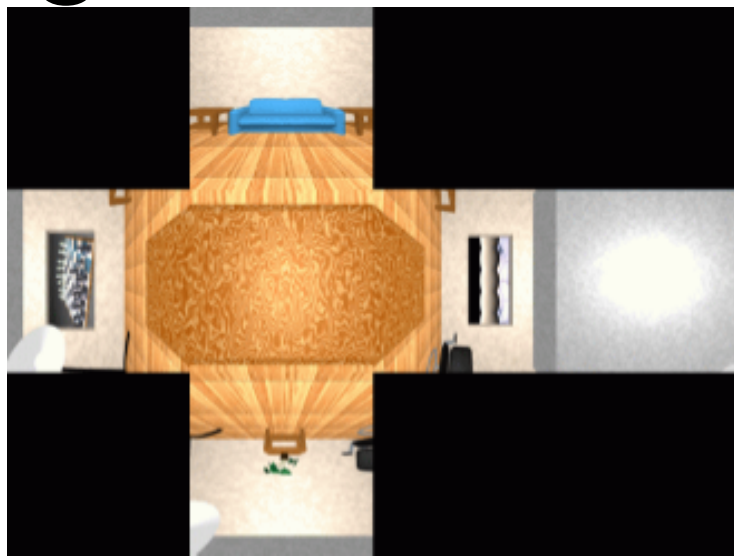
- Simple yet powerful method to generate reflections
- Simulate reflections by using the reflection vector to index a texture map at "infinity".



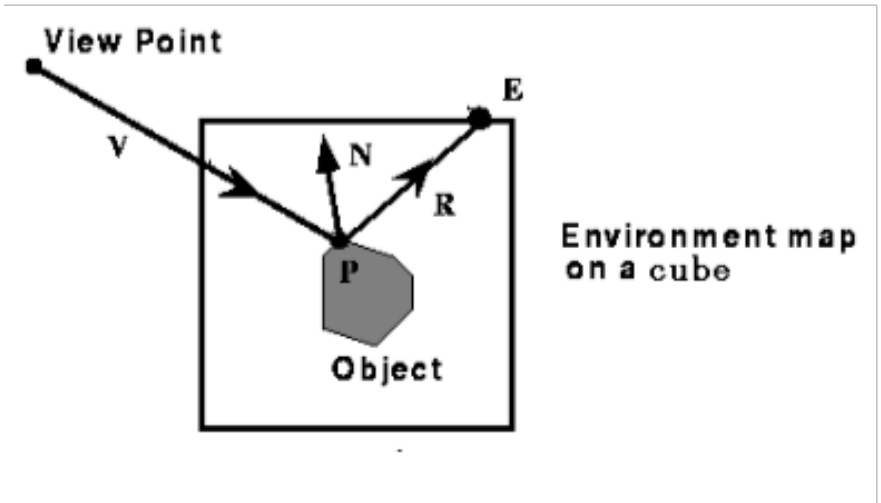


The original environment map was
a sphere [by Jim Blinn ’76]

Cubic Mapping



- The most popular method
- The map resides on the surfaces of a cube around the object
  - align the faces of the cube with the coordinate axes

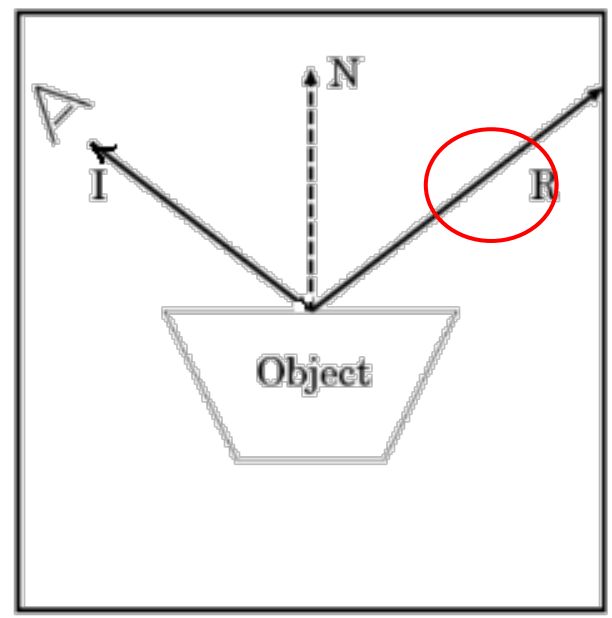


Procedure

During the rasterization, for every pixel,

1. **Calculate the reflection vector R using the camera (incident) vector and the normal vector of the object N**
2. Select the face of the environment map and the pixel on the face according to **R**
3. Colour the pixel with the colour of the environment map

Look up the environment map just using **R**
Do not take into account the 3D position of the reflection point

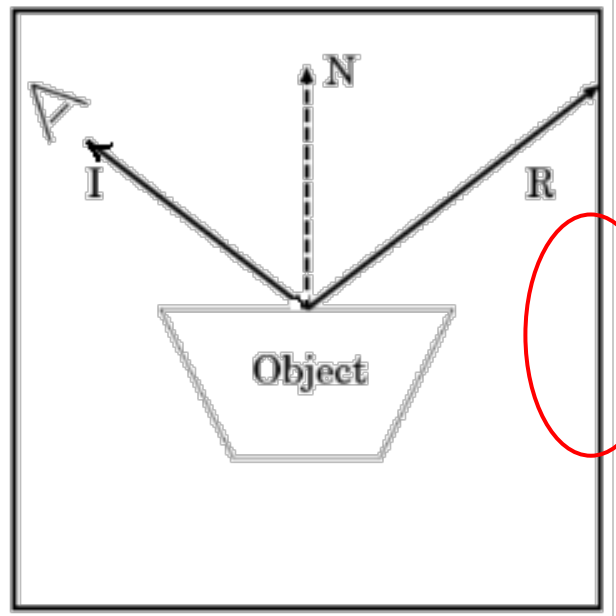


Procedure

During the rasterization, for every pixel,

1. Calculate the reflection vector **R** using the camera (incident) vector and the normal vector of the object **N**
2. **Select the face of the environment map and the pixel on the face according to R**
3. Colour the pixel with the colour of the environment map

Look up the environment map just using **R**
Do not take into account the 3D position of the reflection point

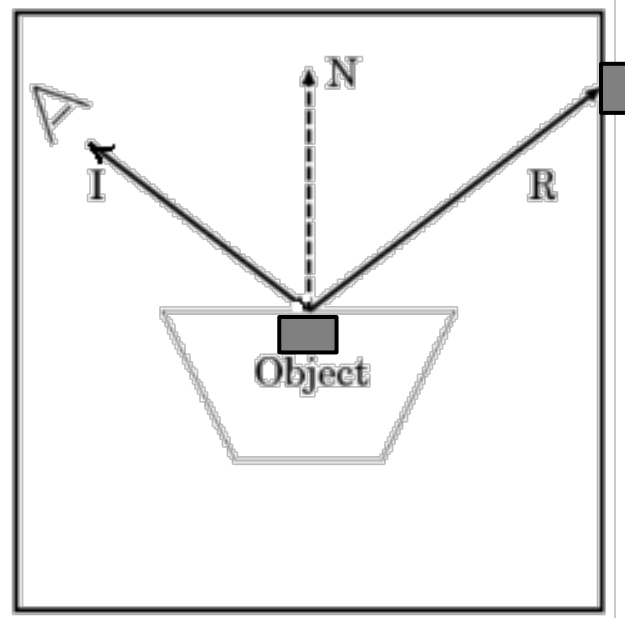


Procedure

During the rasterization, for every pixel,

1. Calculate the reflection vector **R** using the camera (incident) vector and the normal vector of the object **N**
2. Select the face of the environment map and the pixel on the face according to **R**
3. Colour the pixel with the colour of the environment map

Look up the environment map just using **R**
Do not take into account the 3D position of the reflection point

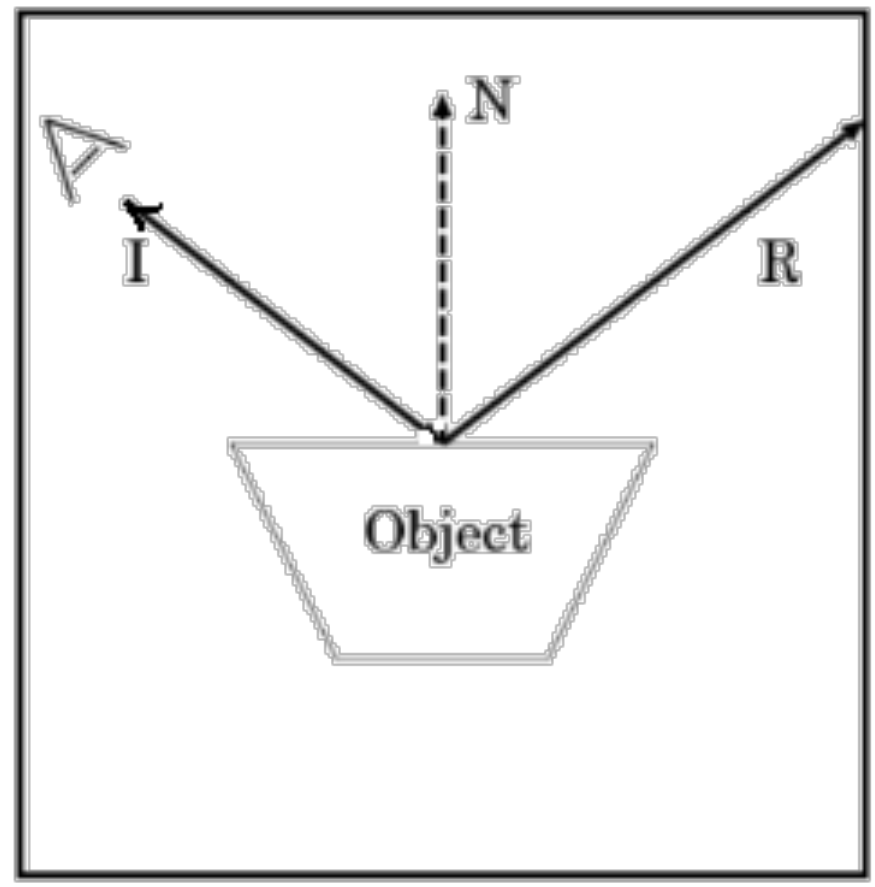


Calculating the reflection vector

- Normal vector of the surface : N
- Eye Ray : I
- Reflection Ray: R
- N,I,R all normalized

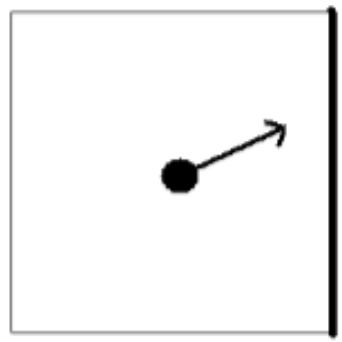
  $$R = 2N(N \cdot I) - I$$

- The texture coordinate is based on the reflection vector
- Assuming the origin of the vector is always in the center of the cube environment map



Indexing Cubic Maps

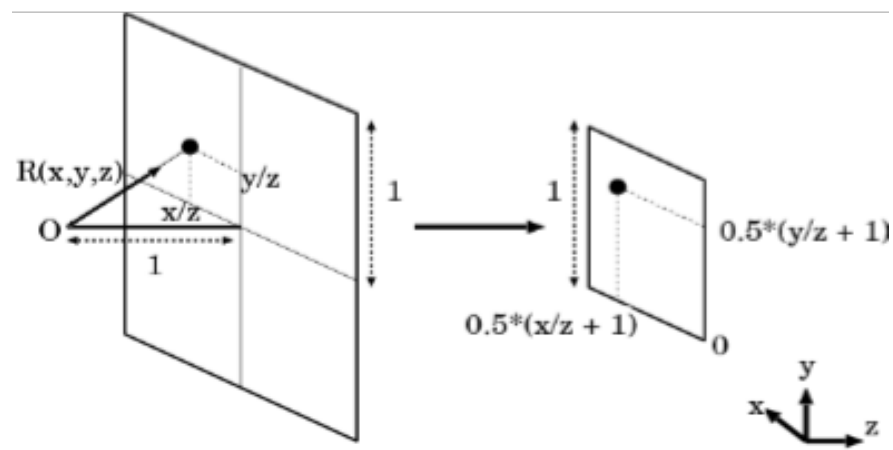
- Assume you have ***R*** and the cube's faces are aligned with the coordinate axes
- How do you decide which face to use?
  - The reflection vector coordinate with the largest magnitude
  - R=(0.3, 0.2, 0.8) -> face in +z direction



Indexing Cubic Maps

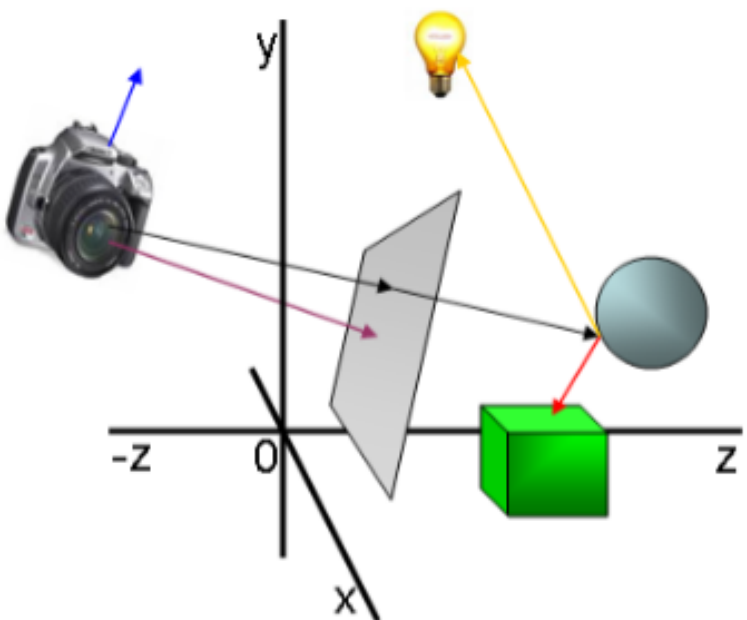
- How do you decide which texture coordinates to use?
- Divide by the coordinate with the largest magnitude
- Now ranging [-1,1]
- Remapped to a value between 0 and 1.

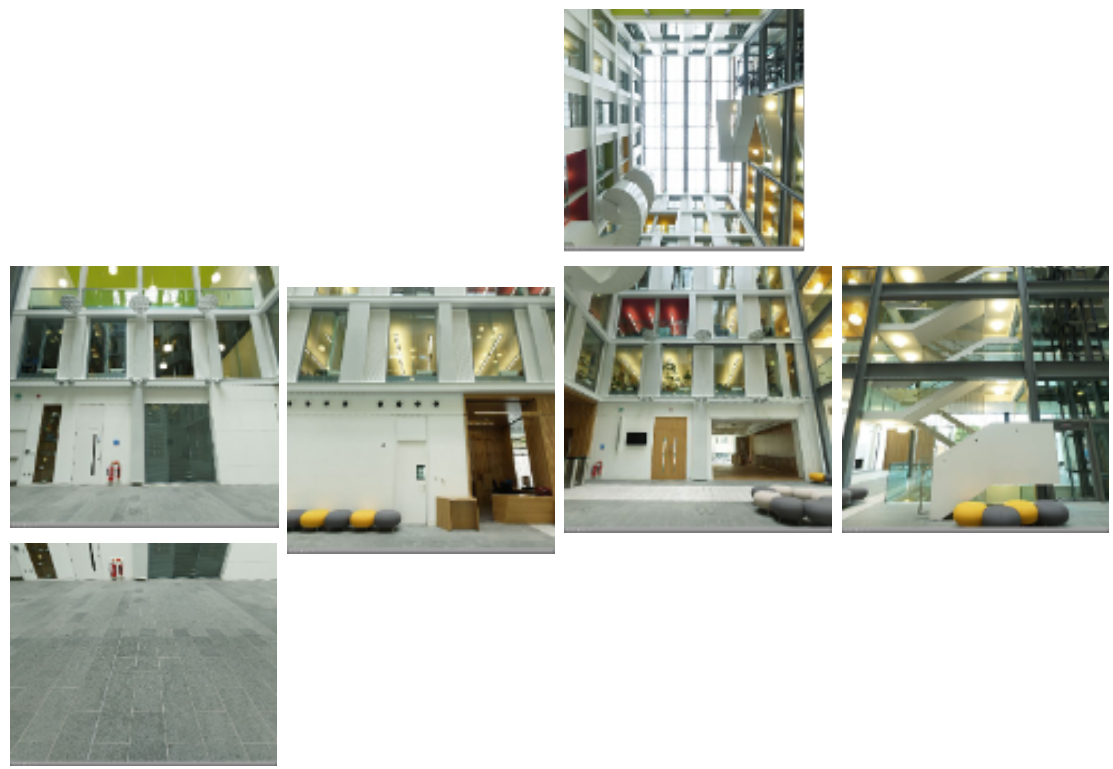
(0.3,0.2,0.8) -> ((0.3/0.8 +1)*0.5, ((0.2/0.8 +1)*0.5) = (0.6875, 0.625)



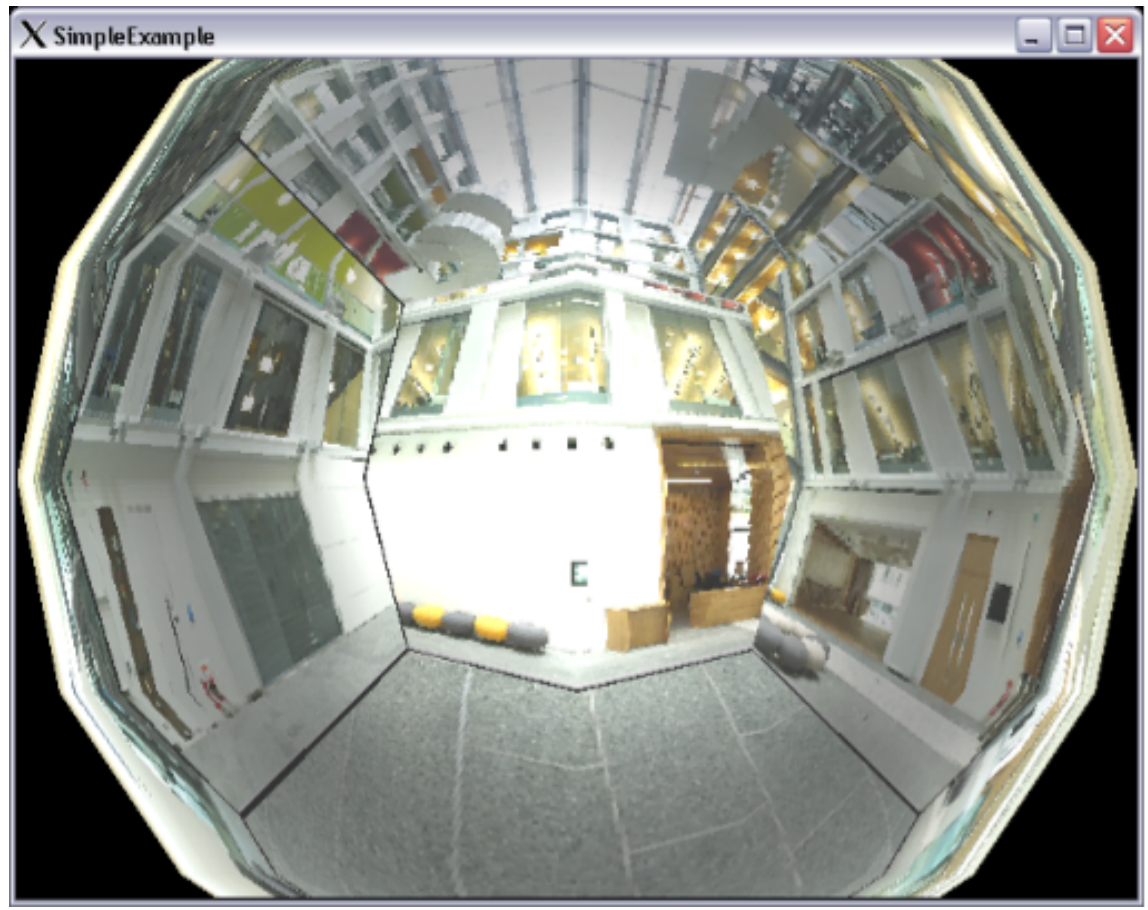
Cubic Mapping: How to make one?

- To generate the map:
- Compute by computer graphics
- Or, take 6 photos of a real environment with a camera in the object's position : much easier





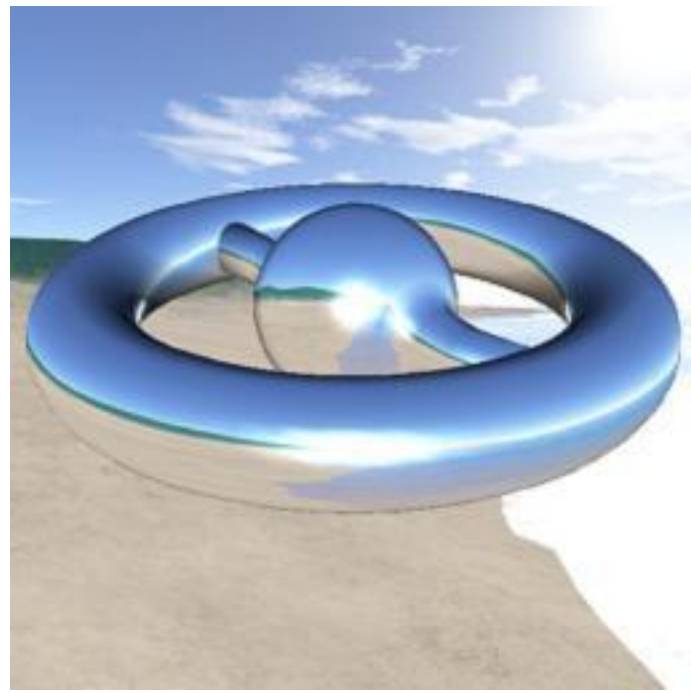
Made from the Forum Images



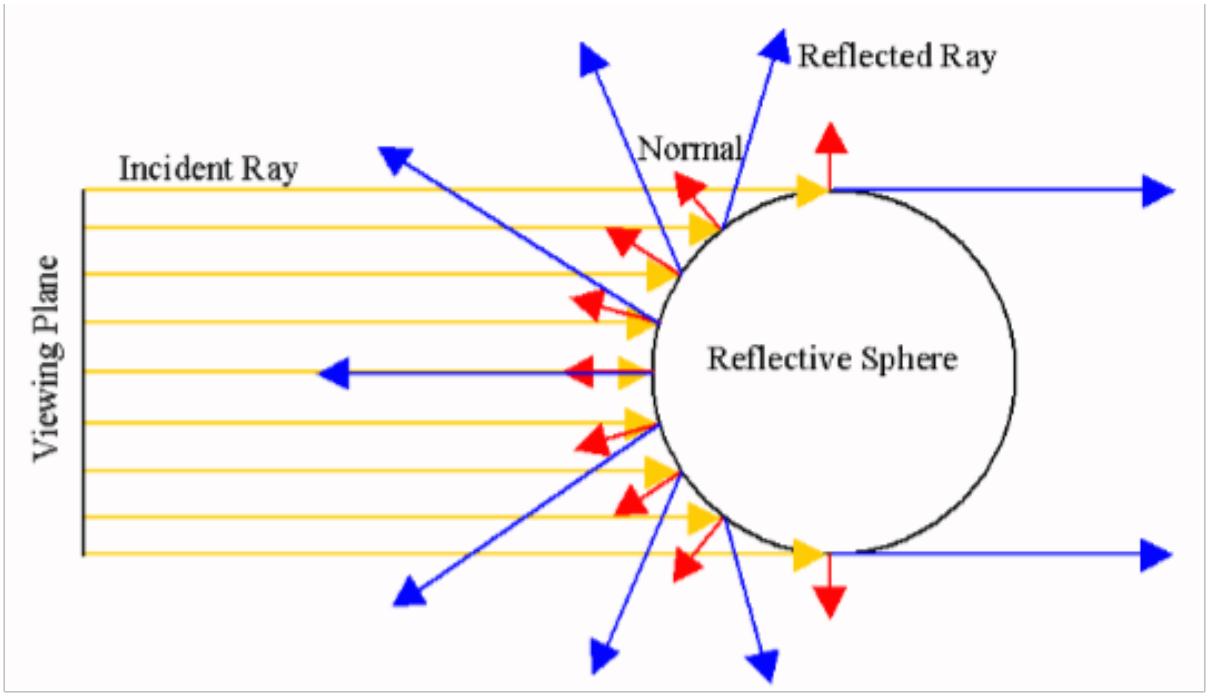
What are the potential problems?

How is it different from rendering the scene more accurately?

What will we miss by environment mapping?

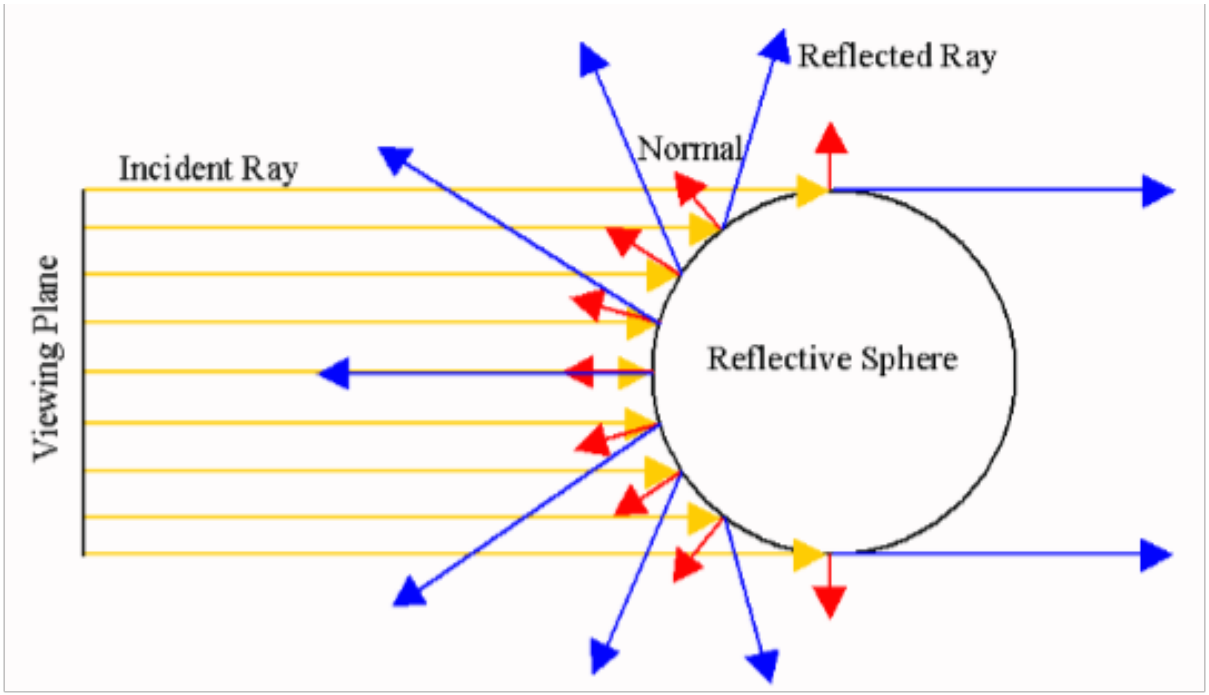


A Sphere Map



- A mapping between the reflection vector and a circular texture
- A mapping between the reflection vector and a circular texture

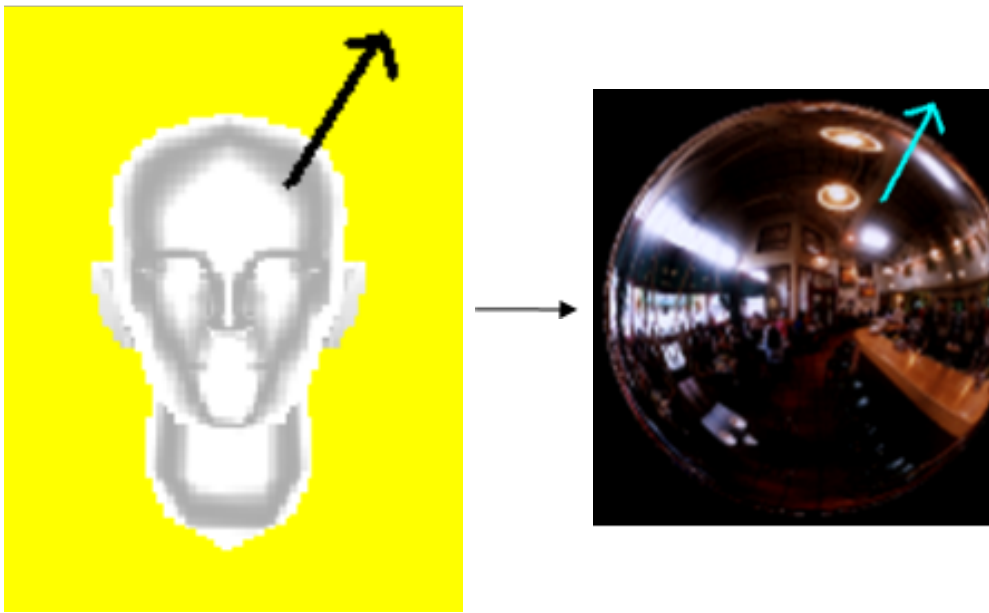
A Sphere Map



The whole environment data is in a single image!

- The resolution near the boundary of the sphere is quite low…

Sphere Map : Procedure



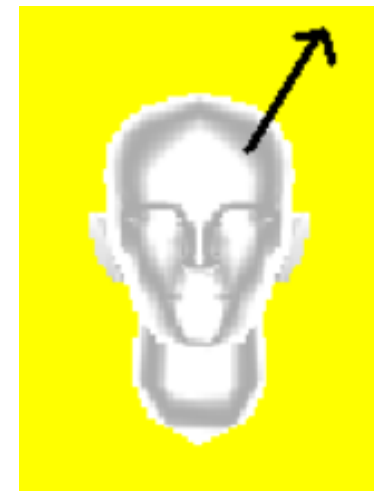
- Compute the reflection vector at the surface of the object
- Find the corresponding texture on the sphere map
- Use the texture to color the surface of the object

Indexing Sphere Maps

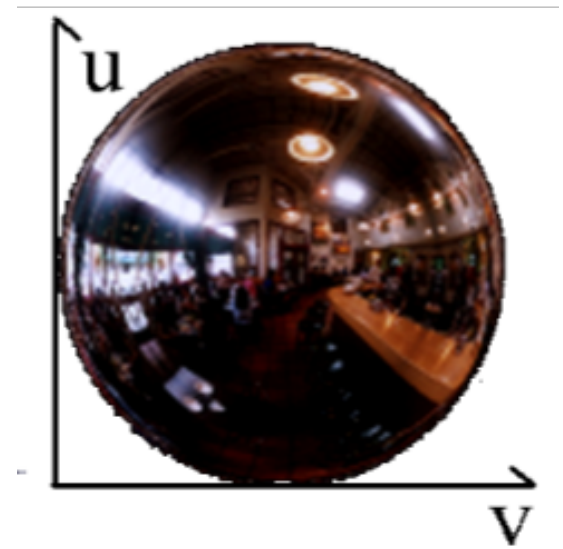
- Given the reflection vector ***R*** (Rx,Ry,Rz)

$$u = \frac{R_x}{2m} + \frac{1}{2}, \quad v = \frac{R_y}{2m} + \frac{1}{2}$$

$$m = \sqrt{R_x^2 + R_y^2 + (R_z + 1)^2}$$

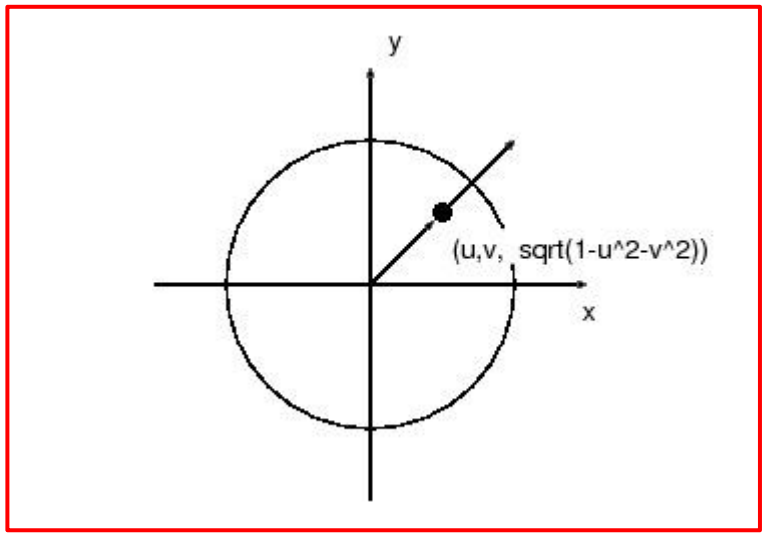


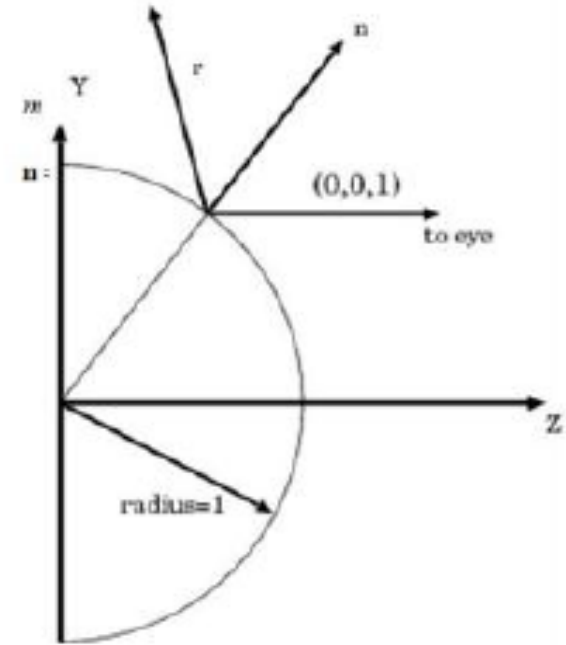
- the (*u,v*) on the spherical map



Indexing in the Sphere Map

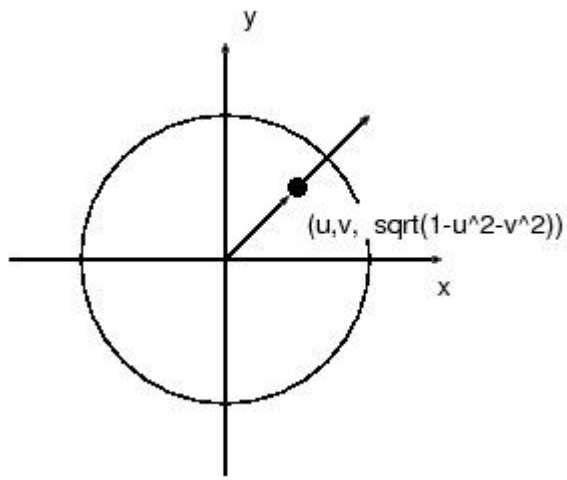
- The x and y elements of the normal vector at point (u,v) ; so N=$(u, v, \sqrt{1-u^2-v^2})$
- The normal vector is the average of the reflection vector (R_x, R_y, R_z) and the camera vector (0,0,1)

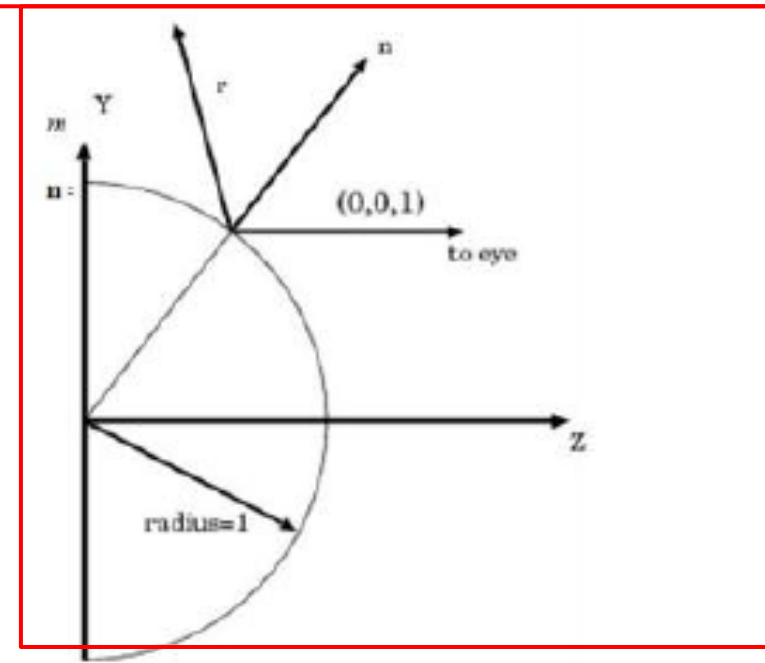




Indexing in the Sphere Map

- The x and y elements of the normal vector at point (u,v) ; so N=$(u, v, \sqrt{1-u^2-v^2})$
- The normal vector is the average of the reflection vector (R_x, R_y, R_z) and the camera vector (0,0,1)





Indexing in the Sphere Map

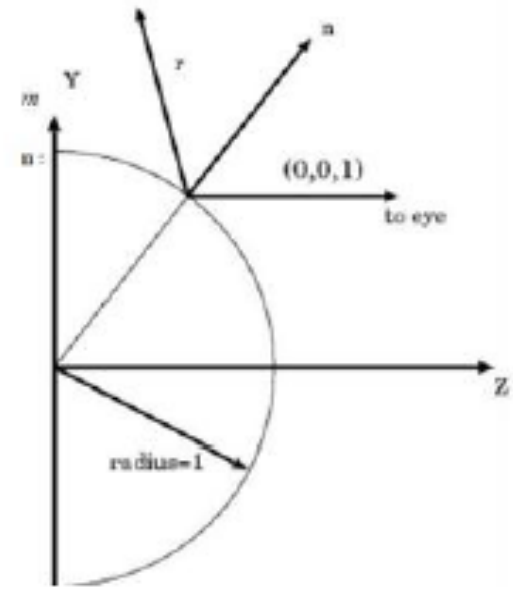
- $N = \left(\frac{R_x}{2m}, \frac{R_y}{2m}, \frac{R_z}{2m} + \frac{1}{2}\right)$

Where $m = \sqrt{R_x^2 + R_y^2 + (R_z + 1)^2}$

- Therefore, given the reflection vector, the index in the sphere map can be computed by
- $(u,v) = \left(\frac{R_x}{2m}, \frac{R_y}{2m}\right)$
- Assuming the center is (1/2,1/2)

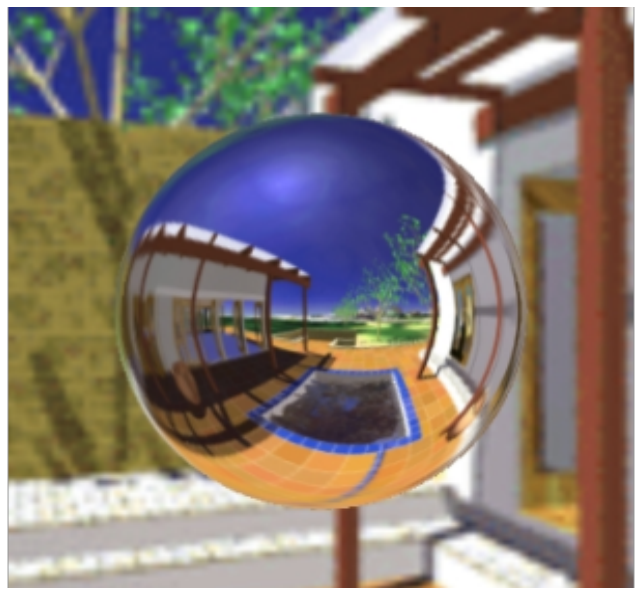
$$u = \frac{R_x}{2m} + \frac{1}{2}, \quad v = \frac{R_y}{2m} + \frac{1}{2}$$

$$m = \sqrt{R_x^2 + R_y^2 + (R_z + 1)^2}$$



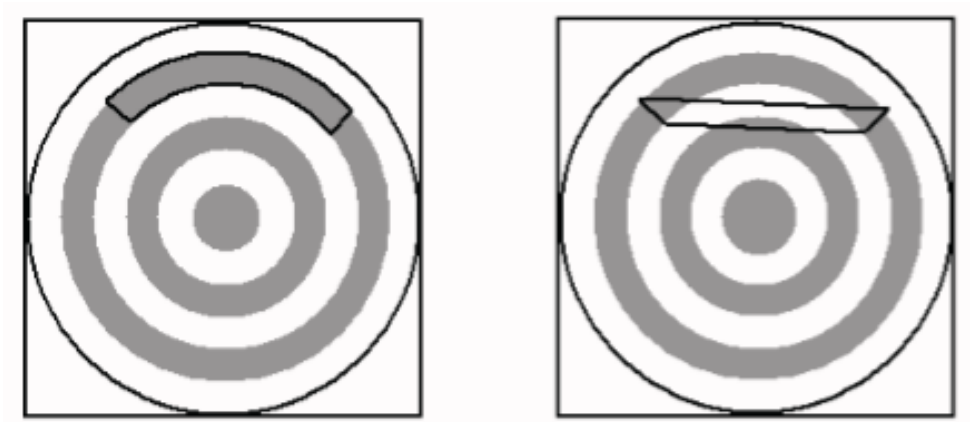
To generate the Sphere Mapping

- Take a photograph of a shiny sphere
- Mapping a cubic environment map onto a sphere
- For synthetic scenes, you can use ray tracing



Issues with the Sphere Mapping

- Cannot change the viewpoint
  - Requires recomputing the sphere map
- Highly non-uniform sampling
- Highly non-linear mapping
- Linear interpolation of texture coordinates picks up the wrong texture pixels
- Do per-pixel sampling or use high resolution polygons

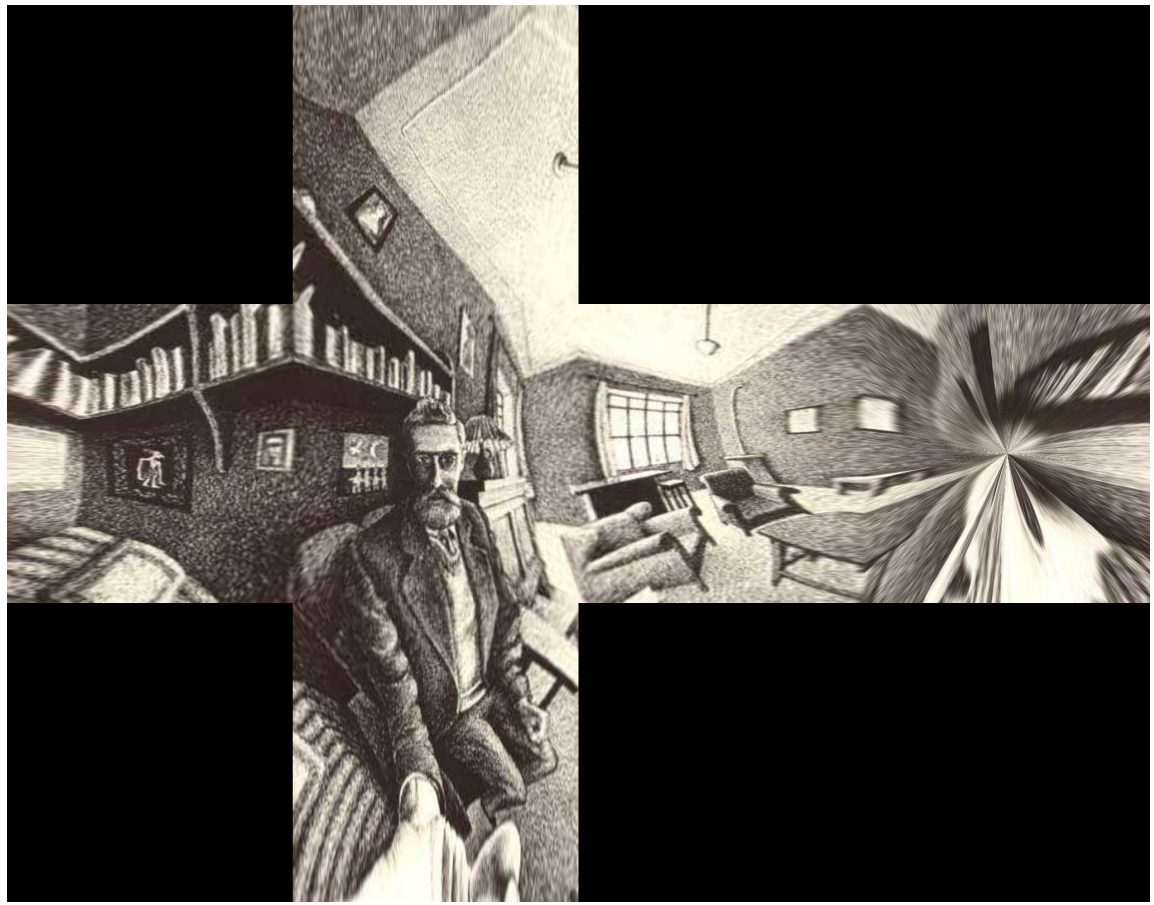


Correct Linear

How can you make the right image from the left image?

Where does middle point on the right image corresponds to?

by Mark VandeWettering



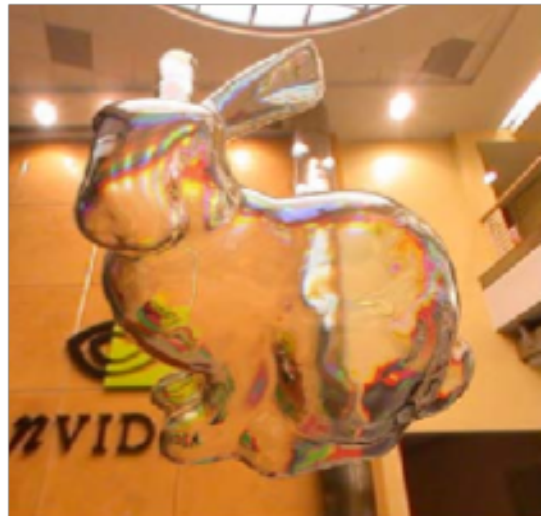
Cons and Pros

How do you compare cube mapping and sphere mapping?

- Advantages of cube mapping?
- Problem of sphere mapping?

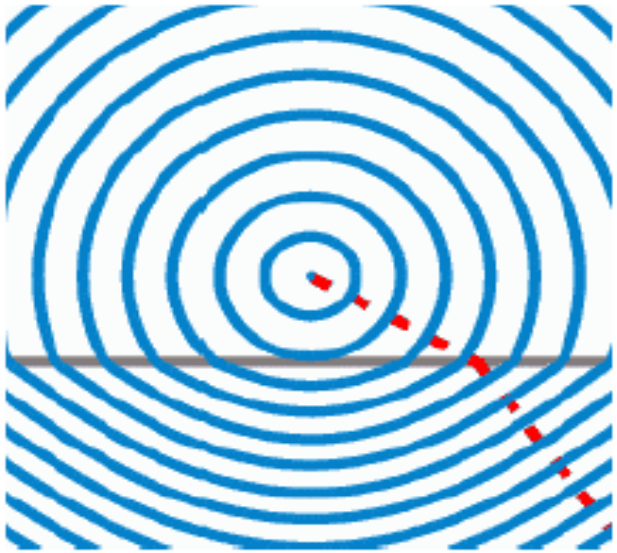
Refractive Environment Mapping

When simulating effects mapping the refracted environment onto translucent materials such as ice or glass, we must use Refractive Environment Mapping

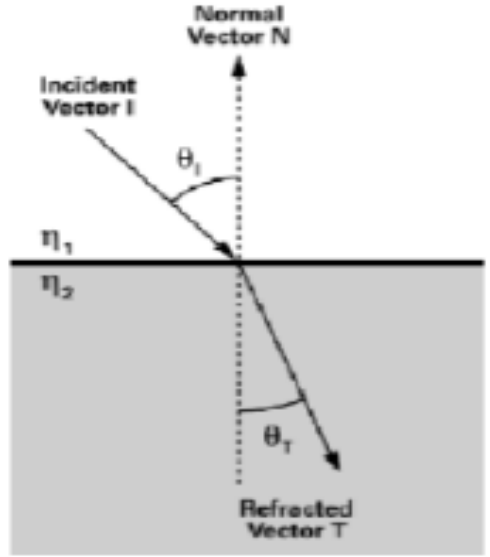


Snell's Law

- Light travels at different speed in different media



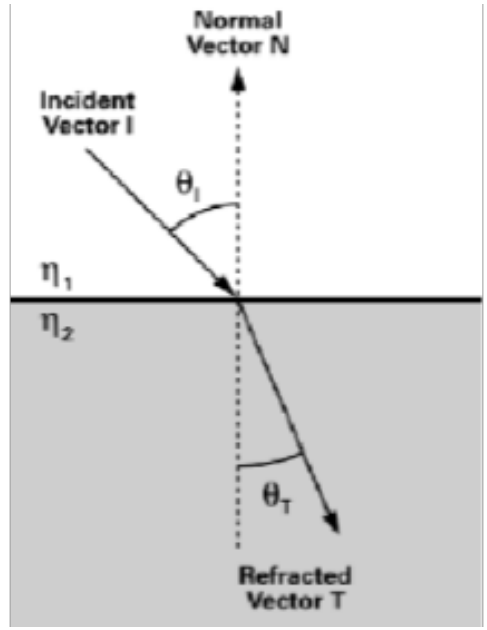
Material	Index of Refraction
Vacuum	1.0
Air	1.0003
Water	1.3333
Glass	1.5
Plastic	1.5
Diamond	2.417



$$\eta_1 \sin\theta_I = \eta_2 \sin\theta_T$$

Snell's Law

- When light passes through a boundary between two materials of different density (air and water, for example), the light's direction changes.
- The direction follows Snell's Law
- We can do environment mapping using the refracted vector **t**



$$\eta_1 \sin\theta_I = \eta_2 \sin\theta_T$$

Material	Index of Refraction
Vacuum	1.0
Air	1.0003
Water	1.3333
Glass	1.5
Plastic	1.5
Diamond	2.417

Snell's Law

- i: incoming vector
- t: refraction vector

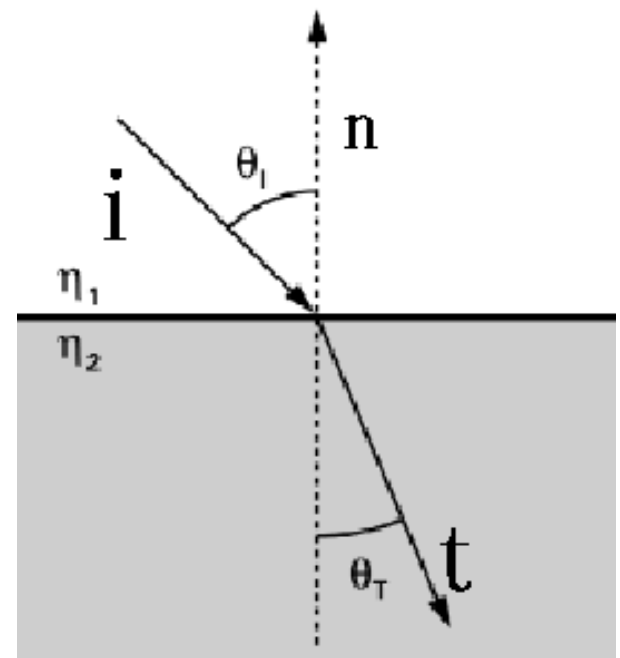
$$\mathbf{t} = r\mathbf{i} + (w - k)\mathbf{n}$$

where

$$r = \frac{n_1}{n_2}$$

$$w = -(\mathbf{i} \bullet \mathbf{n})r,$$

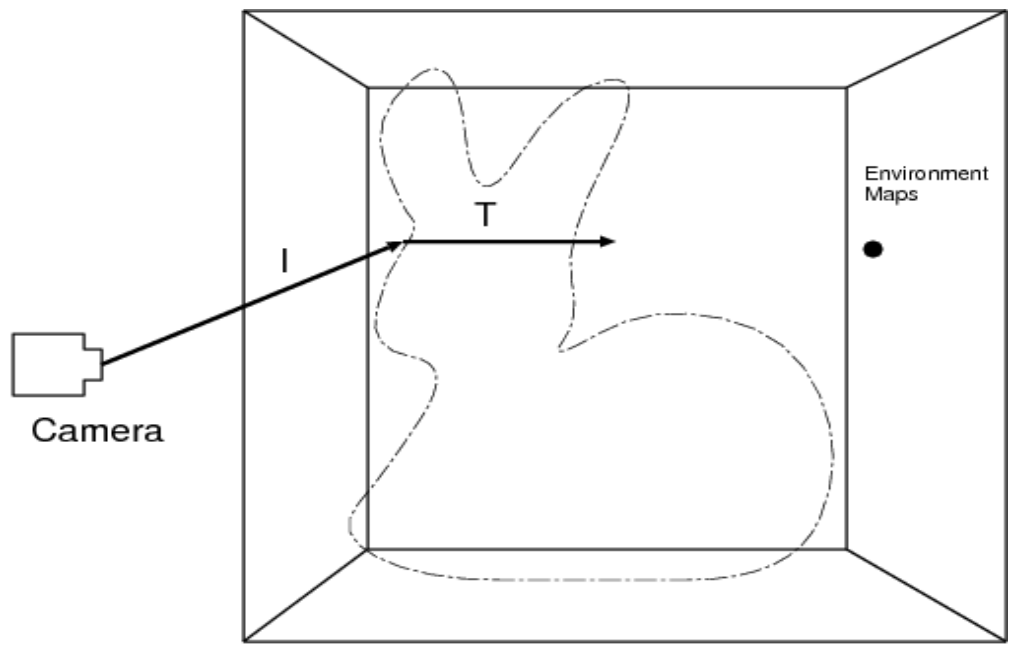
$$k = \sqrt{1 + (w - r)(w + r)}$$



Refractive Environment Mapping

Just use the refraction vector after the first hit as the index to the environment map

- –Costly to compute the second refraction vector
- –Better use cubic mapping - Why?



Summary

Environment mapping is a quick way to simulate the effect of reflecting the world at the surface of a glossy object

- Practical approaches are the cubic mapping and the sphere mapping
- Can also be applied for simulating refraction

Today

Environment Mapping

- Introduction
- Cubic mapping
- Sphere mapping
- refractive mapping

Mirroring

- Introduction
- reflection first
- stencil buffer
- reflection last

Mirroring (Flat Mirrors) : Background

Basic idea: Drawing a scene with mirrors

•Mirrors reflect the world

•A scene with a mirror can be drawn by rendering the world twice

•original scene, and

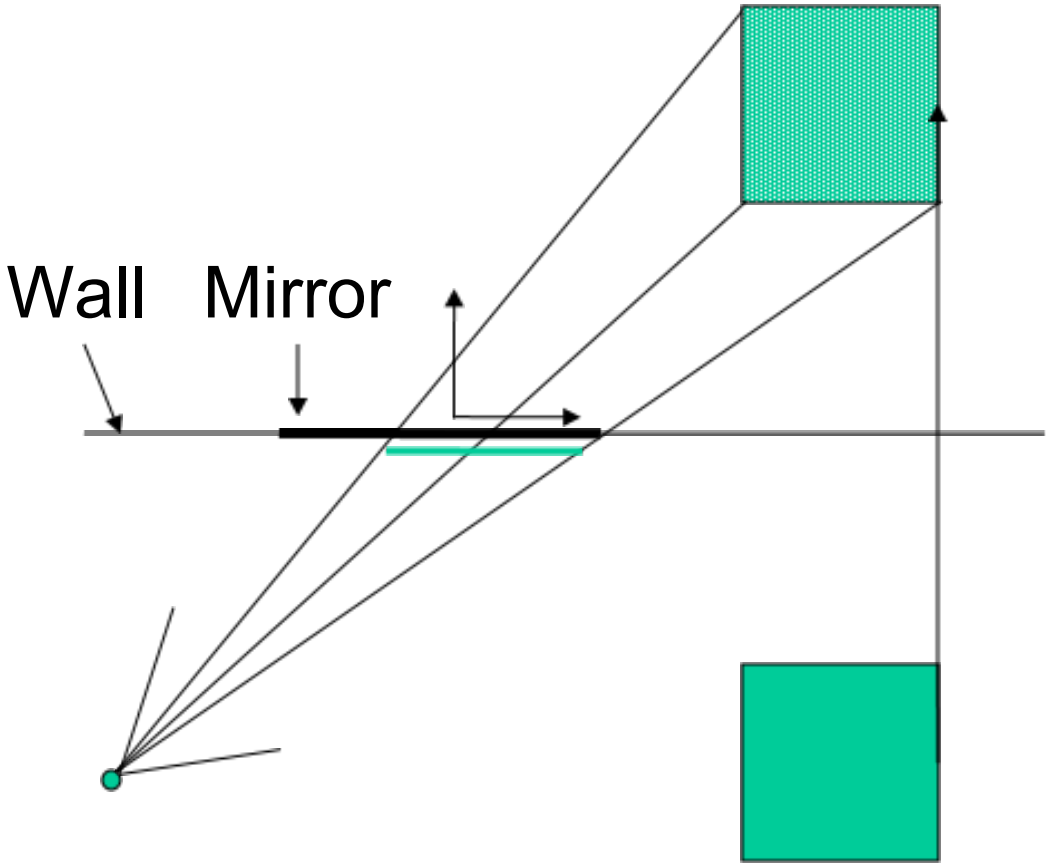
•reflected scene

Mirroring (Flat Mirrors) : Background (2)

Simply rendering the scene twice can result in problems

- The flipped world may appear at area outside the mirror area
- Unless the mirrored world is hidden by the real world
- We can avoid such problems using the “stencil buffer”

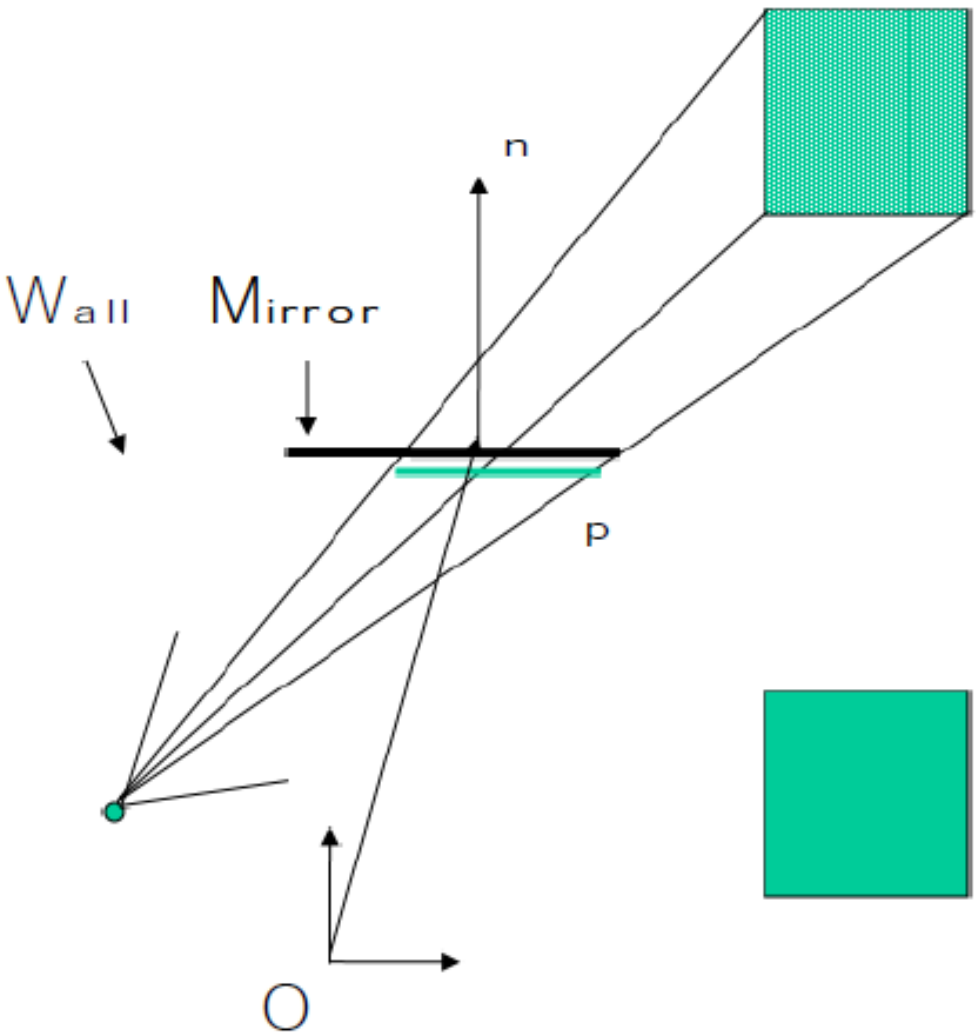
Reflecting Objects



- If the mirror passes through the origin, and is aligned with a coordinate axis, then just negate appropriate coordinate
- For example, if a reflection plane has a normal n= (0,1,0) and passes the origin, the reflected vertices can be obtained by scaling matrix S(1,-1,1)

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Reflecting Objects (2)

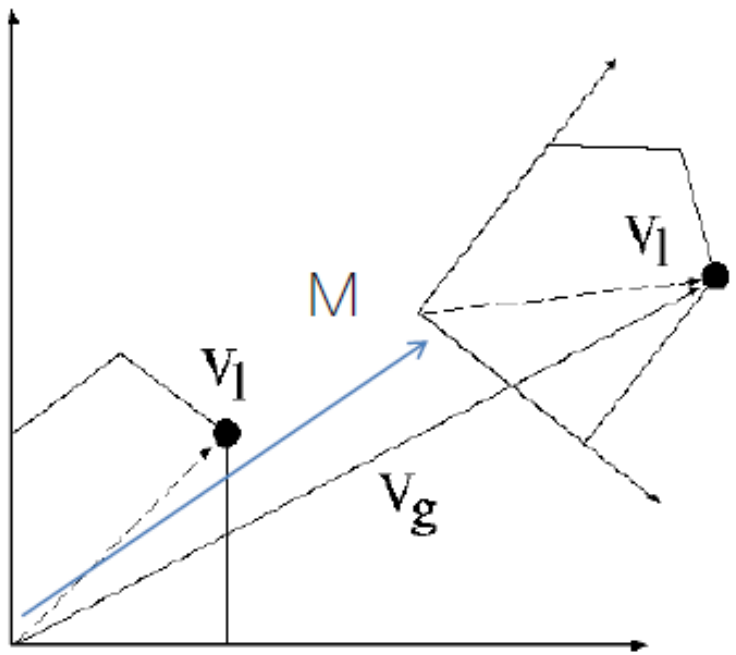


- What if the mirror is not on a plane that passes the origin?
- How do we compute the mirrored world?
- First, we need to compute the location of objects relative to the mirror

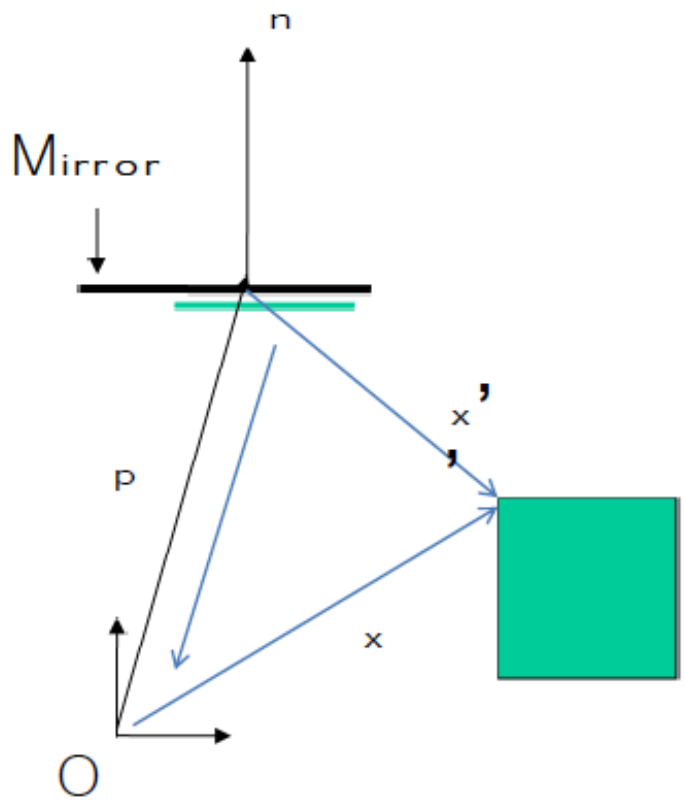
Recap:

Transformations between different coordinate systems

- We can interpret that the transformation matrix is converting the location of vertices between different coordinate systems
- $v_g = M v_l$
- $v_l = M^{-1} v^g$



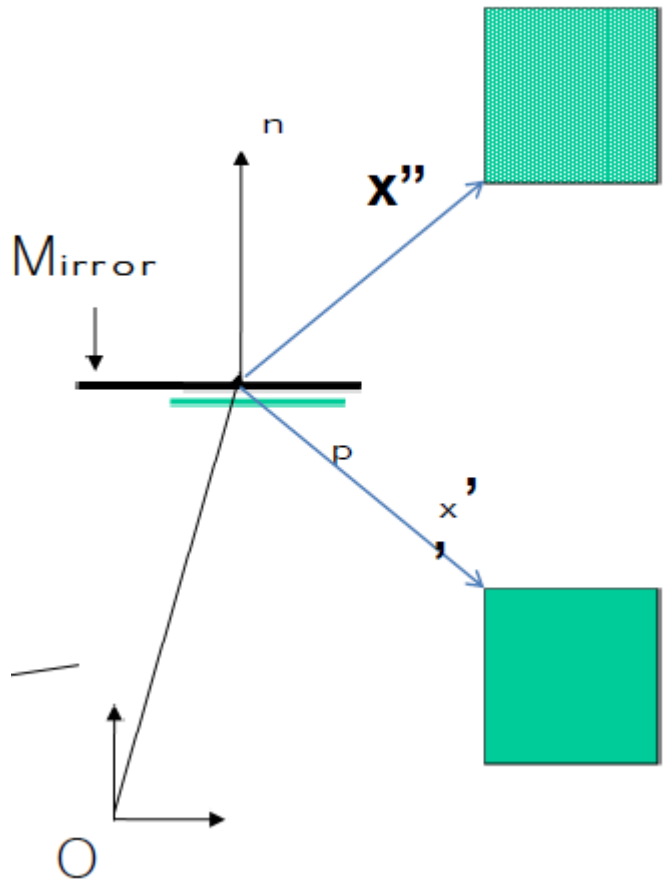
Reflecting Objects (3)



- To know the positions of objects with respect to the mirror coordinate,
- we multiply a transformation matrix from the mirror to the world coordinate to their positions in the world coordinate

$$x' = R(n)^{-1}T(-p)\,x$$

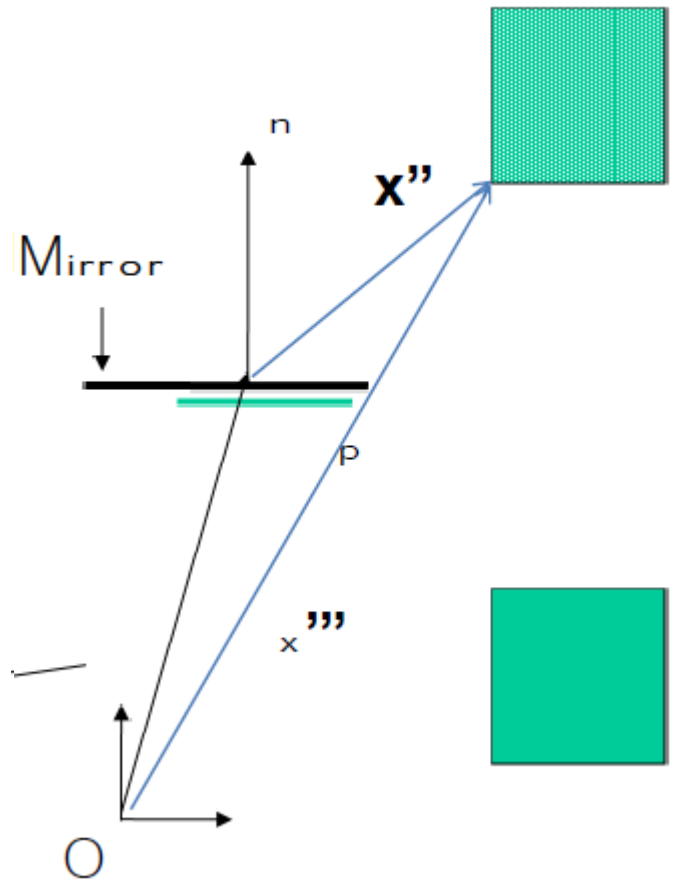
Reflecting Objects (4)



- For finding out the flipped location in the mirror coordinate, we multiply the mirroring matrix

$$x'' = S(1,1,-1)\, x'$$

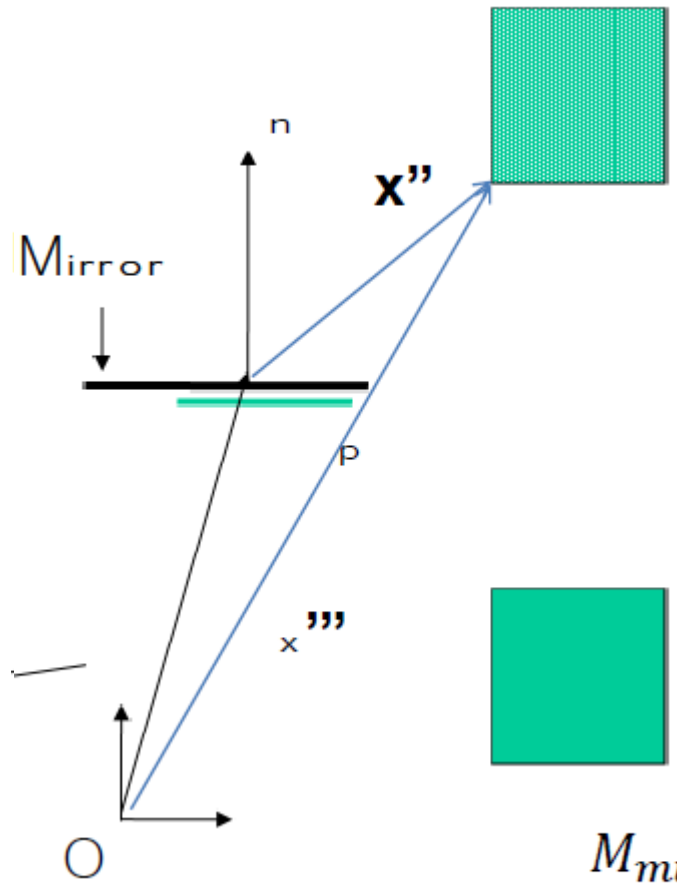
Reflecting Objects (4)



- Now we want to know where the flipped points are with respect to the world origin
- We can multiply the transformation matrix to move from the origin to the mirror to x’’ to know where it is with respect to O

$$x''' = T(p)R(n)\,x''$$

Reflecting Objects (5)



- Altogether

$$x' = R(n)^{-1}T(-p)\ x$$

$$x'' = S(1,1,-1)\ x'$$

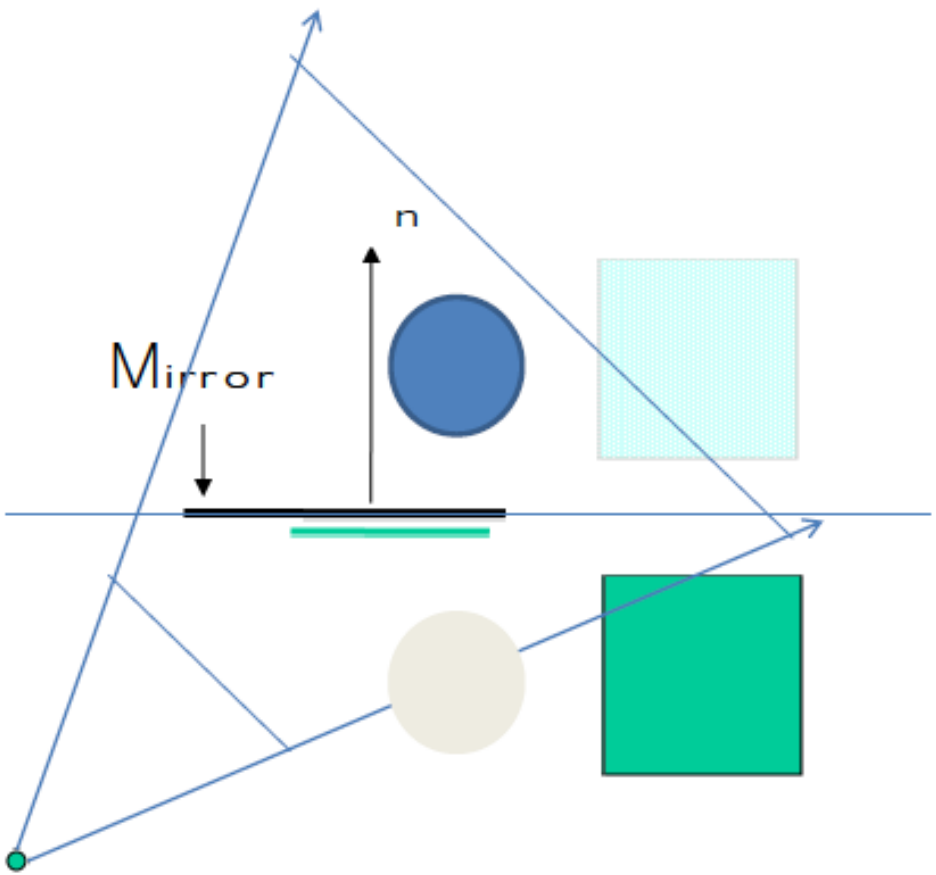
$$x''' = T(p)R(n)\ x''$$

$$x''' = T(p)R(n)S(1,1,-1)R(n)^{-1}T(-p)\ x$$

$$M_{mirrror} = T(p)R(n)S(1,1,-1)R(n)^{-1}T(-p)$$

Avoid drawing the reflected world appearing in the real world

- Specify a user-defined clipping plane
  - Every thing in front of the clipping plane will not be displayed
  - Set it at the plane where the mirror is set



User defined clipping plane

Drawing the mirrored world

Two ways to do it:

1. Draw the mirrored world first, then the real world
   - Only using the depth (Z) buffer
   - Does not work in some cases

2. Draw the real-world first, and then the mirrored world
   - Requires using a stencil buffer

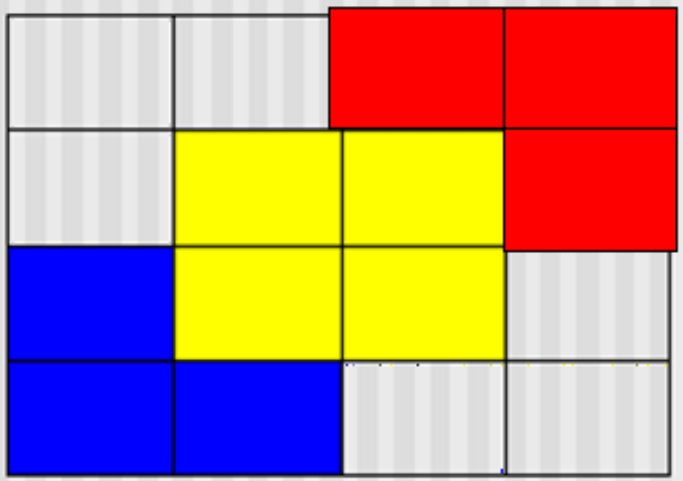
Z-buffer

- One method of hidden surface removal

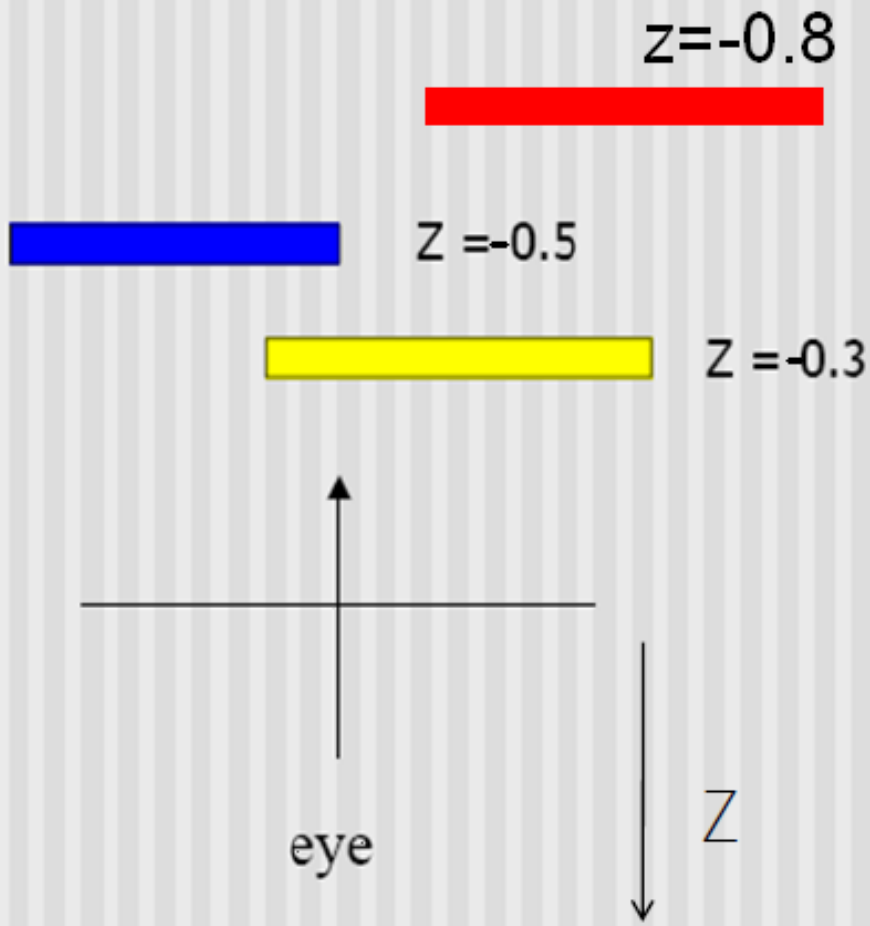
Basic Z-buffer idea: For every input polygon

- For every pixel in the polygon interior, calculate its corresponding z value.
- Compare the depth value with the closest value from a different polygon (largest z) so far
- Paint the pixel (filling in the color buffer) with the color of the polygon if it is closer

Z buffer example



Correct Final image



Top View

Z buffer example

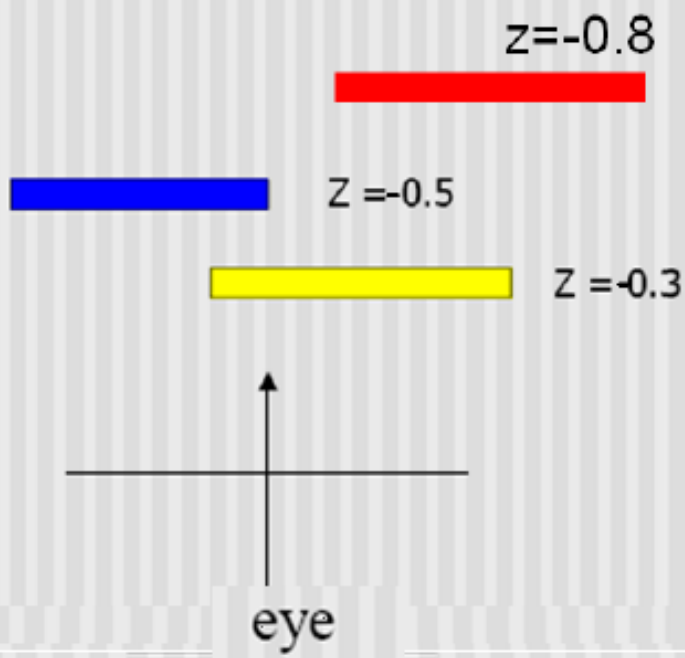
Step 1: Initialize the depth buffer

-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0

Z buffer example

Step 2: Draw the blue polygon (assuming the program draws blue polyon first – the order does not affect the final result any way).

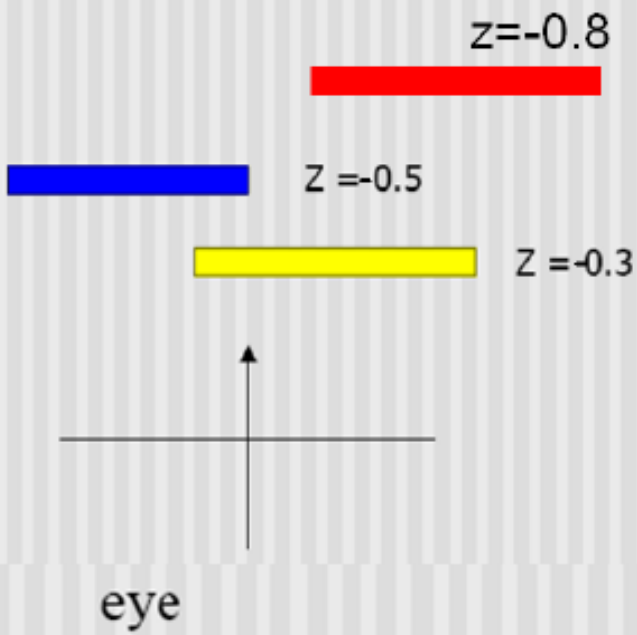
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-0.5	-0.5	-1.0	-1.0
-0.5	-0.5	-1.0	-1.0



Z buffer example

Step 3: Draw the yellow polygon

-1.0	-1.0	-1.0	-1.0
-1.0	-0.3	-0.3	-1.0
-0.5	-0.3	-0.3	-1.0
-0.5	-0.5	-1.0	-1.0

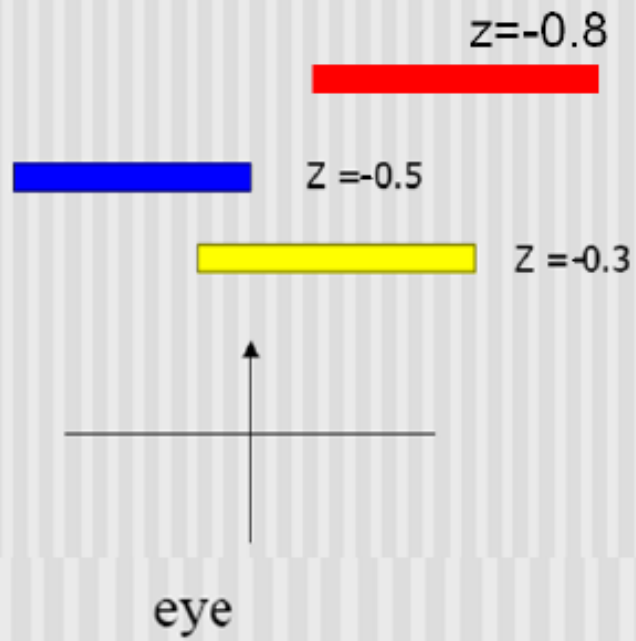


If the depth value is larger than that in the z-buffer, the pixel is coloured and value in the z-buffer is updated

Z buffer example

Step 4: Draw the red polygon

-1.0	-1.0	-0.8	-0.8
-1.0	-0.3	-0.3	-0.8
-0.5	-0.3	-0.3	-1.0
-0.5	-0.5	-1.0	-1.0



If the depth value is larger than that in the z-buffer, the pixel is coloured and value in the z-buffer is updated

Rendering Reflected First (Using the depth buffer(Z-buffer))

First pass: Render the reflected scene without mirror, depth test on

Second pass:

- Disable the color buffer, and render the mirror polygon (to not draw over the reflected scene, but setting the Z-buffer on)
- Now the Z buffer of the mirror region is set to the mirror's surface

Third Pass:

- Enable the color buffer again
- Render the original scene, without the mirror
- Depth buffer stops from writing over things in mirror

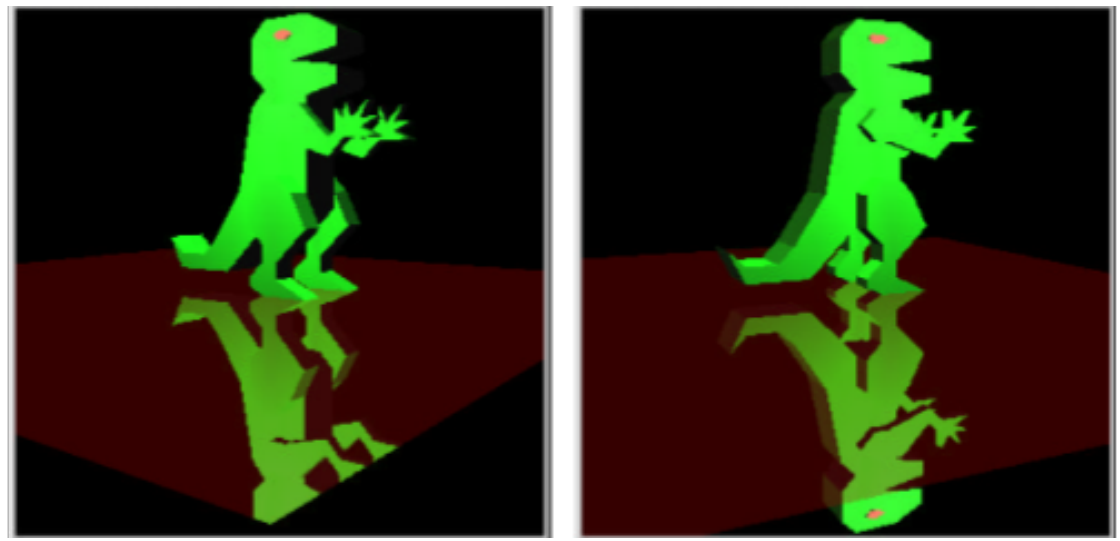
Reflected Scene First (issues)

Objects behind the mirror cause problems:

- The reflected area outside the mirror region is just overwritten by the objects in the front
- unless there is a wall, they will remain visible

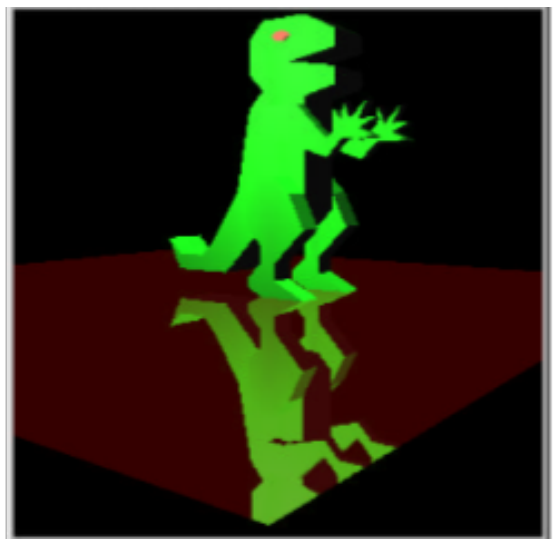
Doesn’t do:

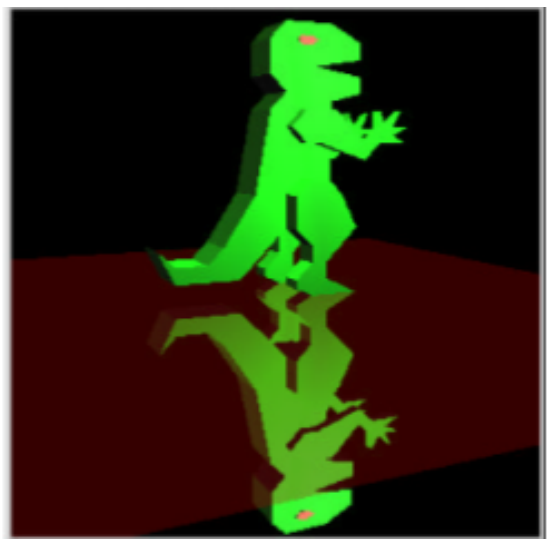
- Reflections of mirrors in mirrors (recursive reflections)
- Multiple mirrors in one scene (that aren’t seen in each other)



Using the Stencil Buffer to Created Scenes with Mirrors

The stencil buffer can help to stop drawing outside the mirror region





We need to use the “Stencil Buffer”

- The stencil buffer acts like a paint stencil - it lets some fragments through but not others
- It stores multi-bit values
- You specify two things:
  - The test that controls which fragments get through
  - The operations to perform on the buffer when the test passes or fails

Reflection Example

Normal first, reflected area next

First pass:

- Render the scene without the mirror

For each mirror

Second pass:

- Clear the stencil, disable the write to the colour buffer, render the mirror, setting the stencil to 1 if the depth test passes

The stencil buffer after the second pass

Third pass:

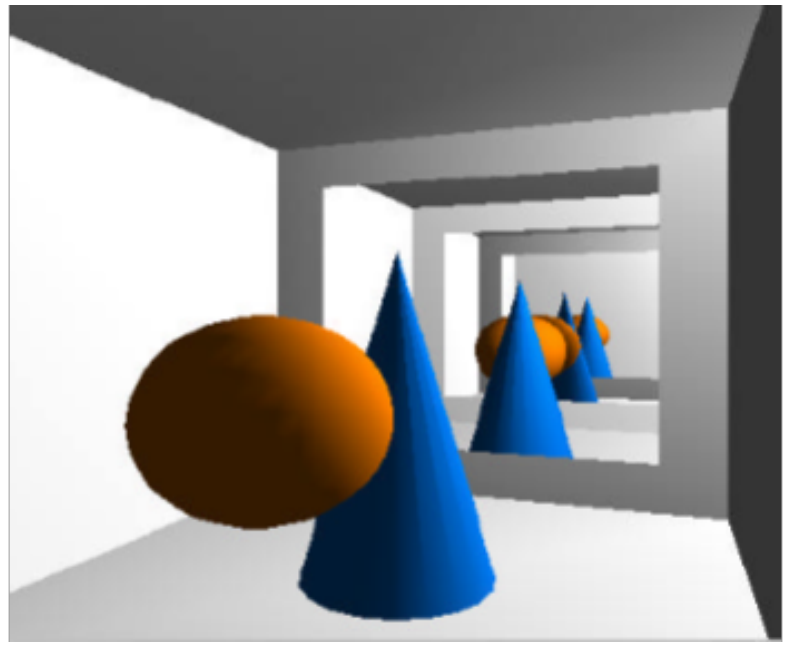
- Clear the depth buffer with the stencil active, passing things inside the mirror only
- Reflect the world and draw using the stencil test. Only things seen in the mirror will be drawn
- Combine it with the scene made during the first pass

Rendering the mirrored scene into the stencil active area

Multiple mirrors

Can manage multiple mirrors

- Render normal view, then do other passes for each mirror
- A recursive formulation exists for mirrors that see other mirrors
- After rendering the reflected area inside the mirror surface, render the mirrors inside the mirror surface, and so on



Conclusion and Summary

- Environment mapping
  - cubic mapping
  - spherical mapping
  - refraction mapping
- Mirroring
  - Flipping the world
  - Zbuffer
  - Stencil buffer

Readings

- Foley 16.5-6
- Real-time Rendering 2, Chapter 5.7, 6.10

Reference

http://brainwagon.org/2002/12/05/fun-with-environment-maps/