CSE 681

Texture Mapping

Why Textures?

- How can we model this scene with polygons?
  - Lots of detail means lots of polygons to render
  - 100’s, 1000’s, Millions of polygons!
  - Modeling is very difficult and cumbersome

Why Textures?

- Render a single polygon with a picture of a brick wall mapped to it

The Quest for Visual Realism

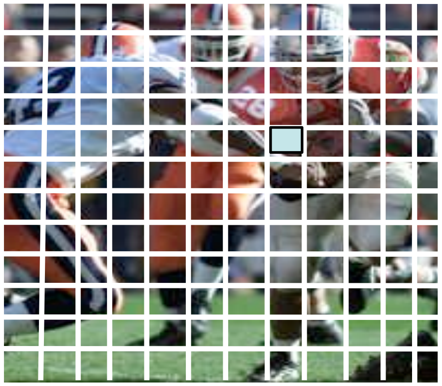
Textures

- Phong lighting just won't do
  - Plastic, rubber, or metallic looking objects

- Add surface detail with real world patterns and images
  - We get details at a *low* cost

- Very useful in games … a billion dollar industry
  - Provides realism at a low cost
  - Graphics hardware vendors work hard to optimize

Terminology



- **Texture**: An array of values
  - 2D (most common), 1D, and 3D
  - Color, alpha, depth, and even normals
- **Texel**: A single array element
- **Texture Mapping**: The process of relating texture to geometry

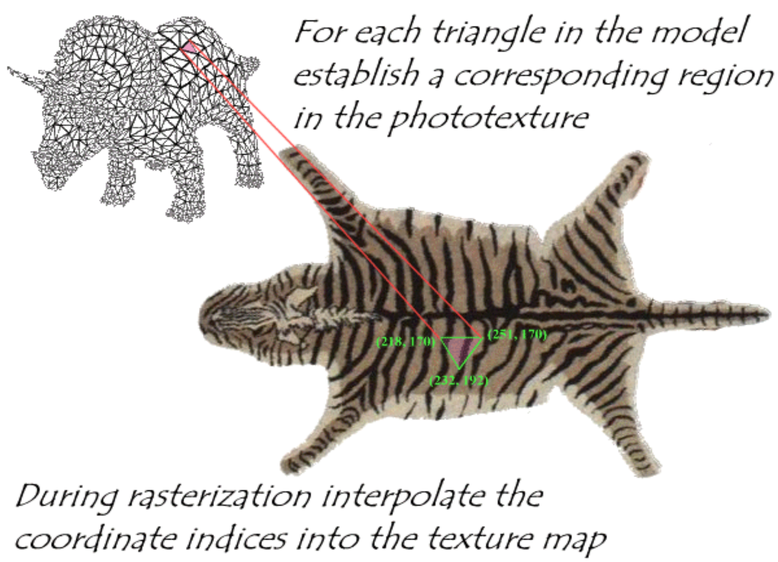
Texture Sources

1. Pixel maps (bitmaps)
   - Load from an image file: gif, jpg, tiff, ppm, etc.

2. Procedural textures
   - Program generated texel values

Texture Mapping

- How do we apply a texture onto an object?
  - Construct a mapping between the texture and object
  - Use the texture to lookup surface attributes

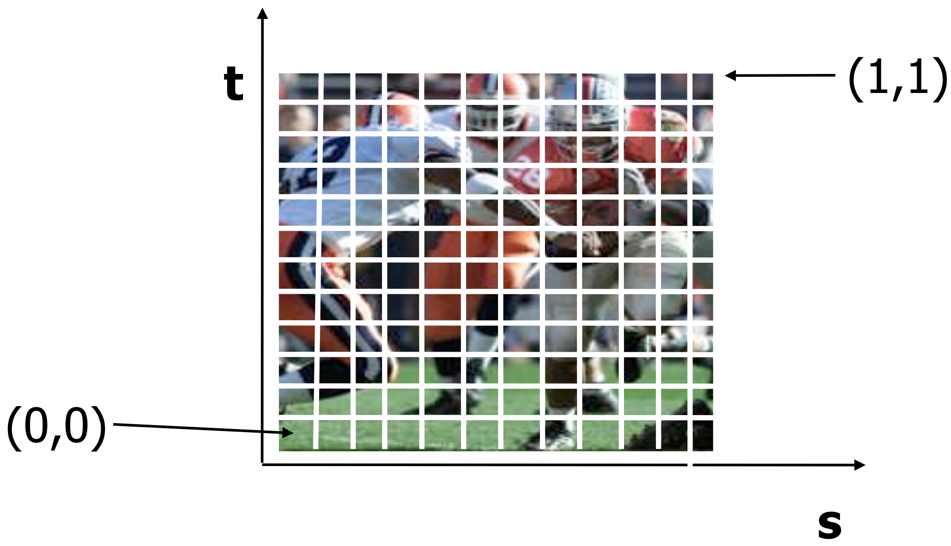


Texture Mapping

- Problems
  - The texture and object are in two different spaces
  - Where in the rendering pipeline do we specify this mapping?
    - Object or world space?
    - Map onto untransformed surfaces
  - Texture filtering: A point on the surface maps to a location between texels in the texture

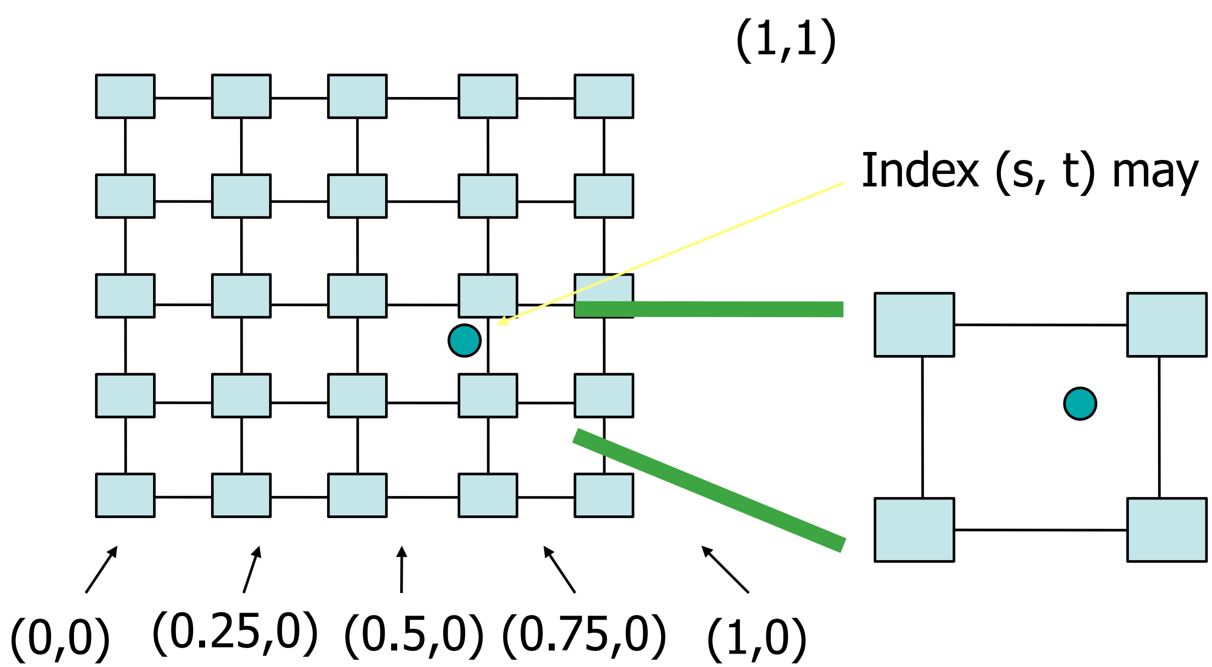
Texture Space

- A texture is defined in a <u>normalized space</u>
  – 2D textures: (s, t) $\in$ [0 … 1, 0 … 1]



Texture Value Lookup/Filtering

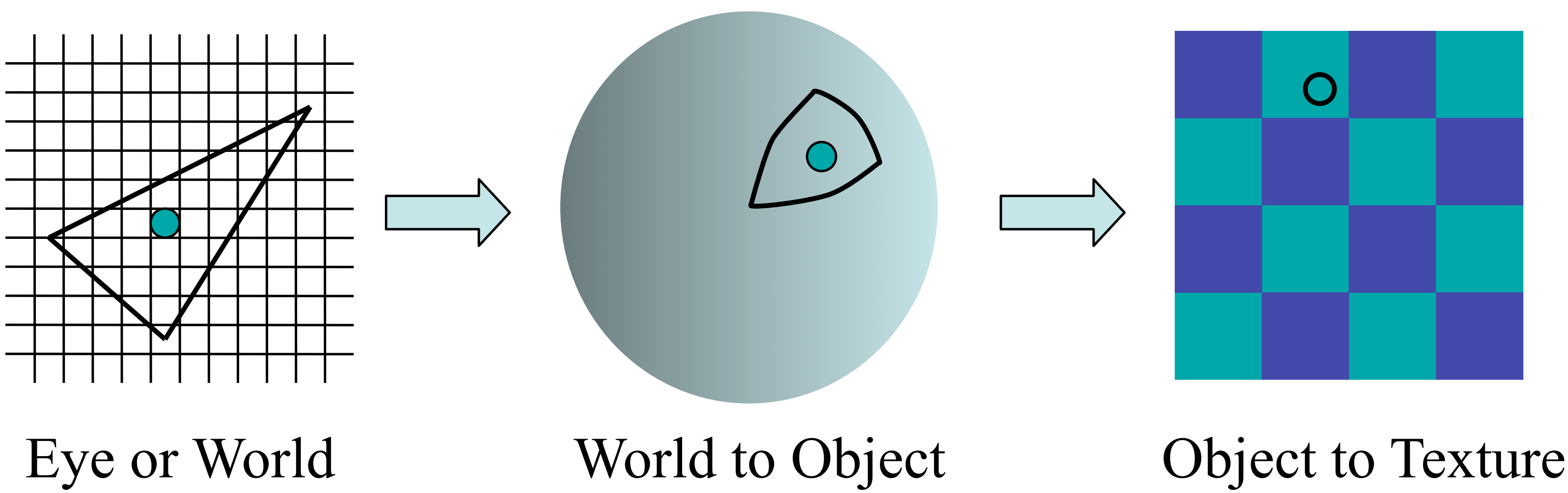
- Normalized space is continuous but the texture is a discrete array
  - Texel values are located on a cartesian grid



A) Nearest neighbor
B) Linear Interpolation
C) Other filters

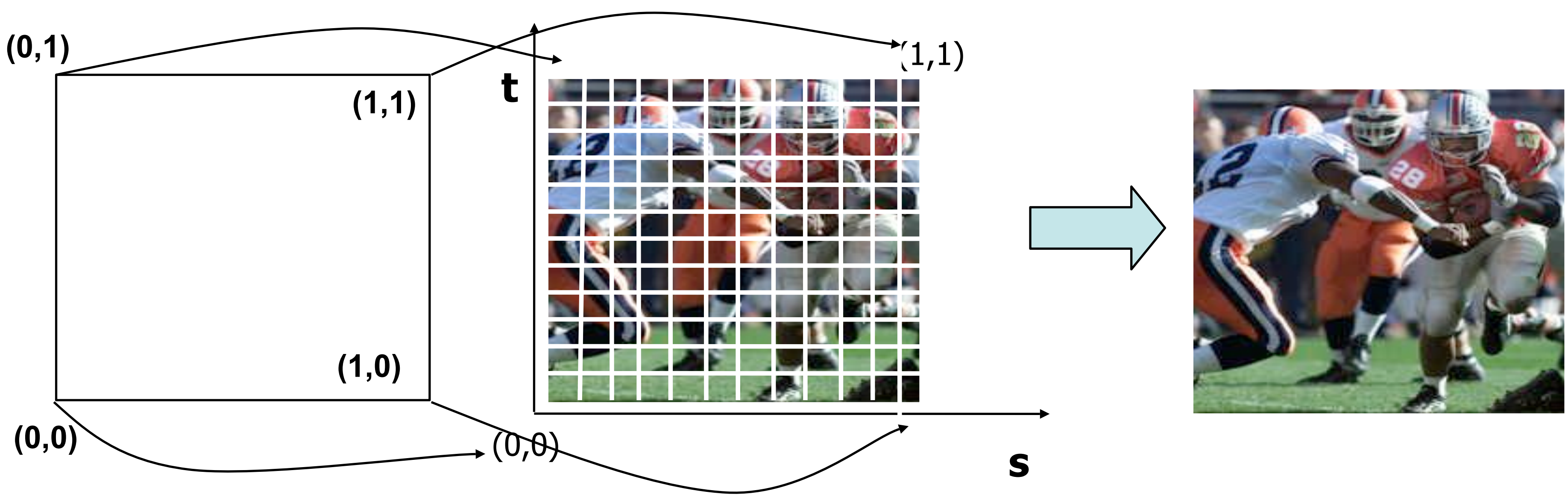
Raytracing a Textured Object

- Shoot ray
  1. Map the intersection point of the visible surface to object space
  2. Map to texture space
  3. Filter the texture
  4. Determine pixel color with retrieved texture information

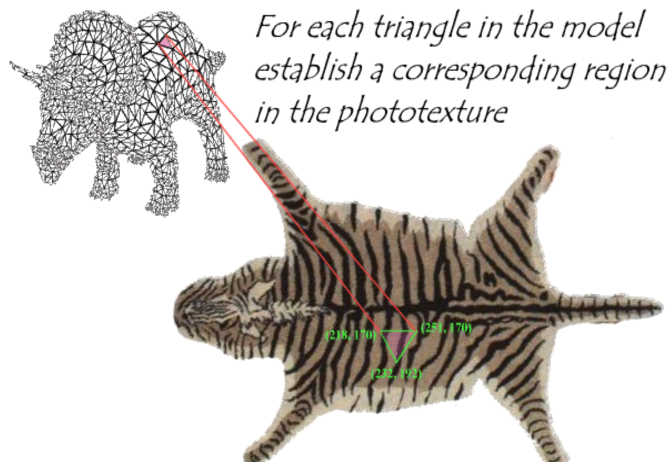


Map 2D Textures To Objects

- Define mapping between object and texture spaces
  - For example, a simple quad in object space is easy!
- Akin to wall papering or gift wrapping



Mapping 2D Textures To Objects

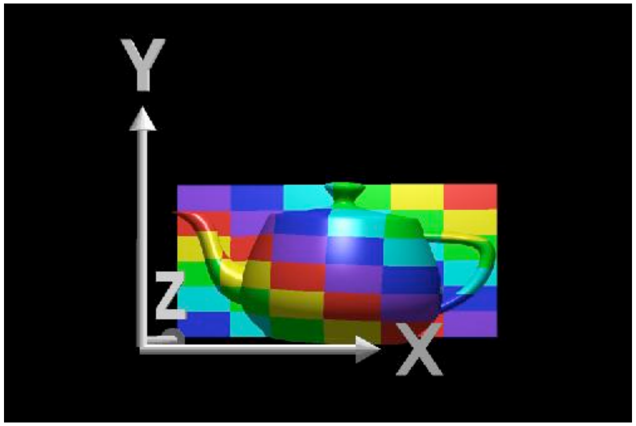


During rasterization interpolate the coordinate indices into the texture map

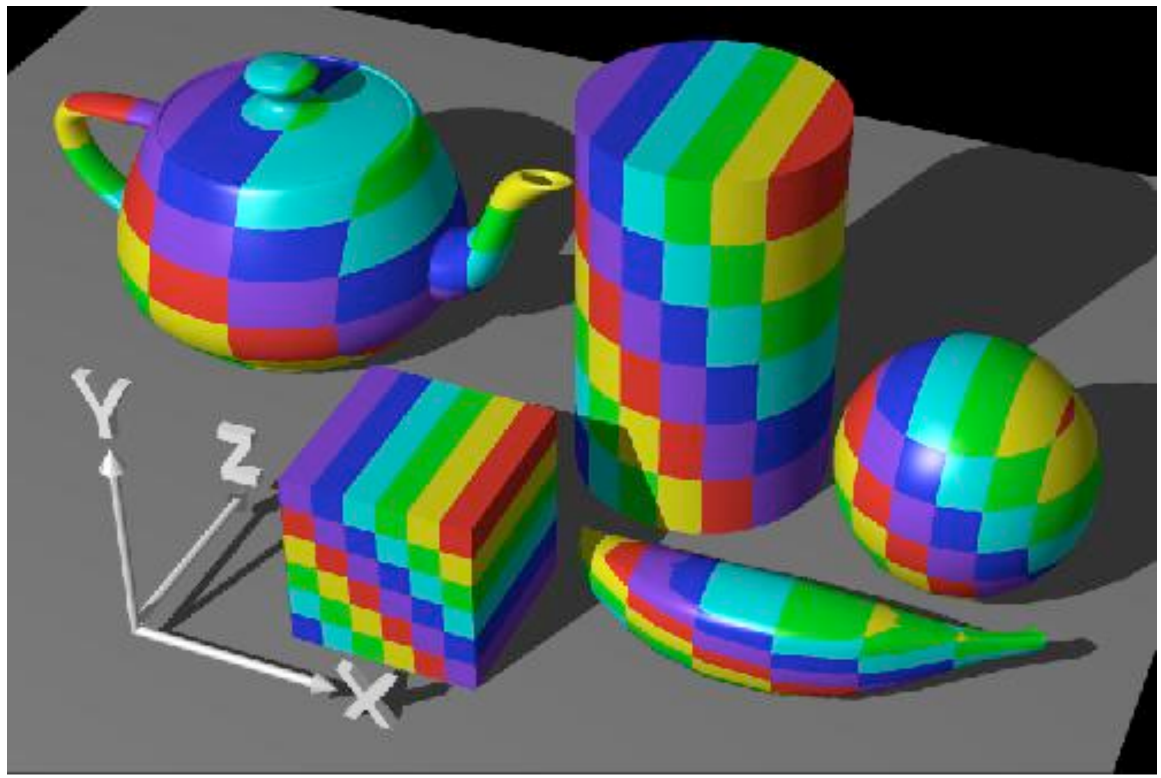
2D Texture Mapping Approaches

- Intermediate Mapping
  - Map the texture onto a simple intermediate surface
  - Map the intermediate surface to the final object
- Intermediate objects
  - Plane
  - Sphere
  - Cylinder
  - Cube

Planar Mapping

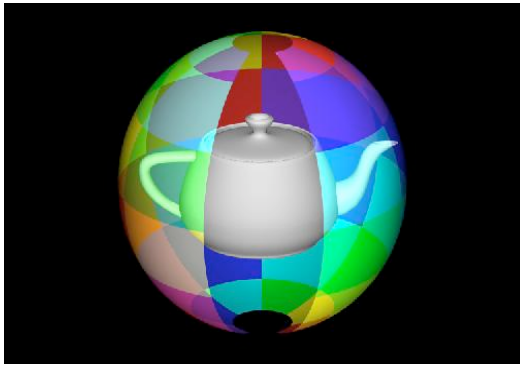


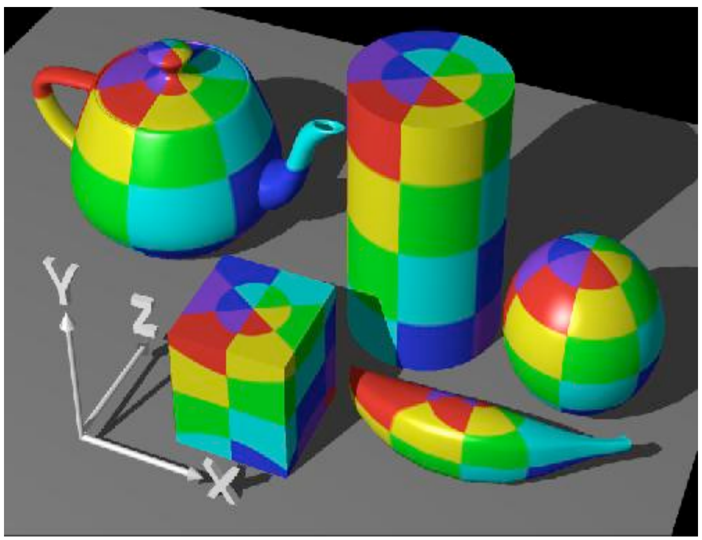
Project to an axial plane,
e.g. drop z coord $(s,t) = (x, y)$

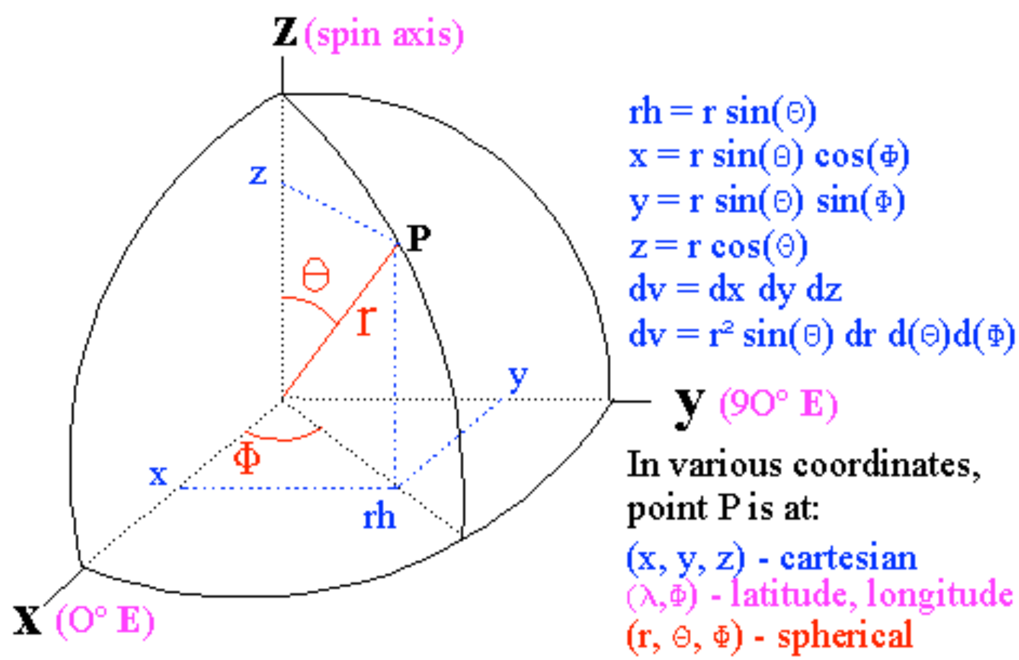


Spherical Mapping

Given a point (x,y,z), convert it to spherical coordinate coordinates (theta,phi)

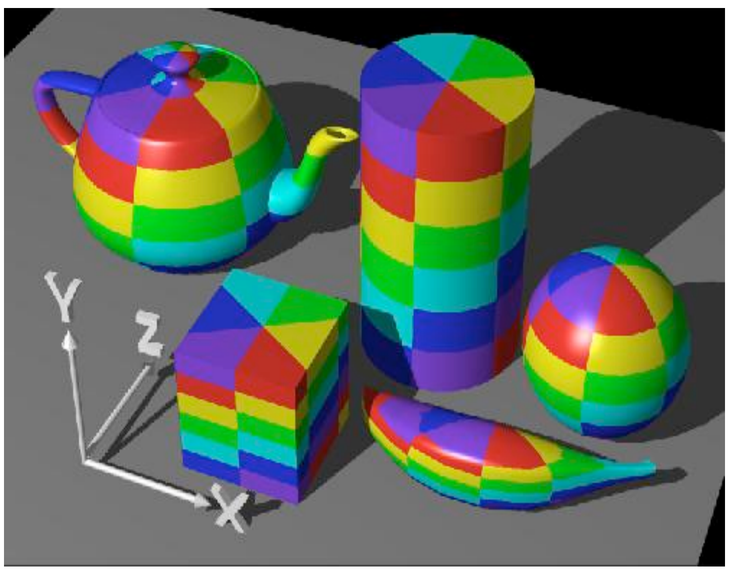






Cylindrical Mapping

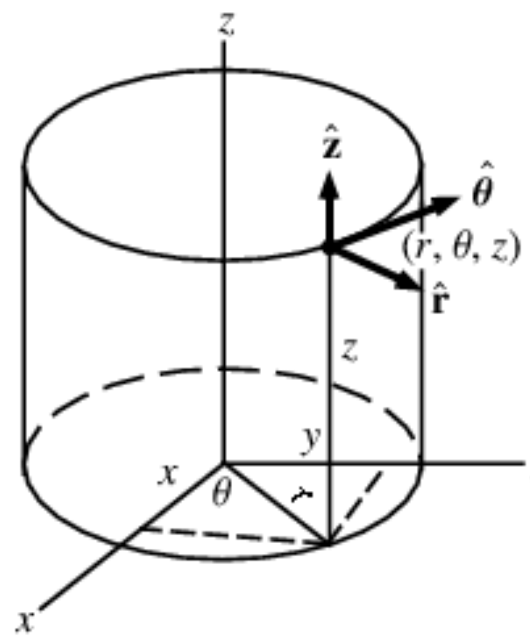
Given a point (x,y,z), convert it to cylindrical coordinates (r, theta, z) and use (theta,z) as the 2D texture coordinates



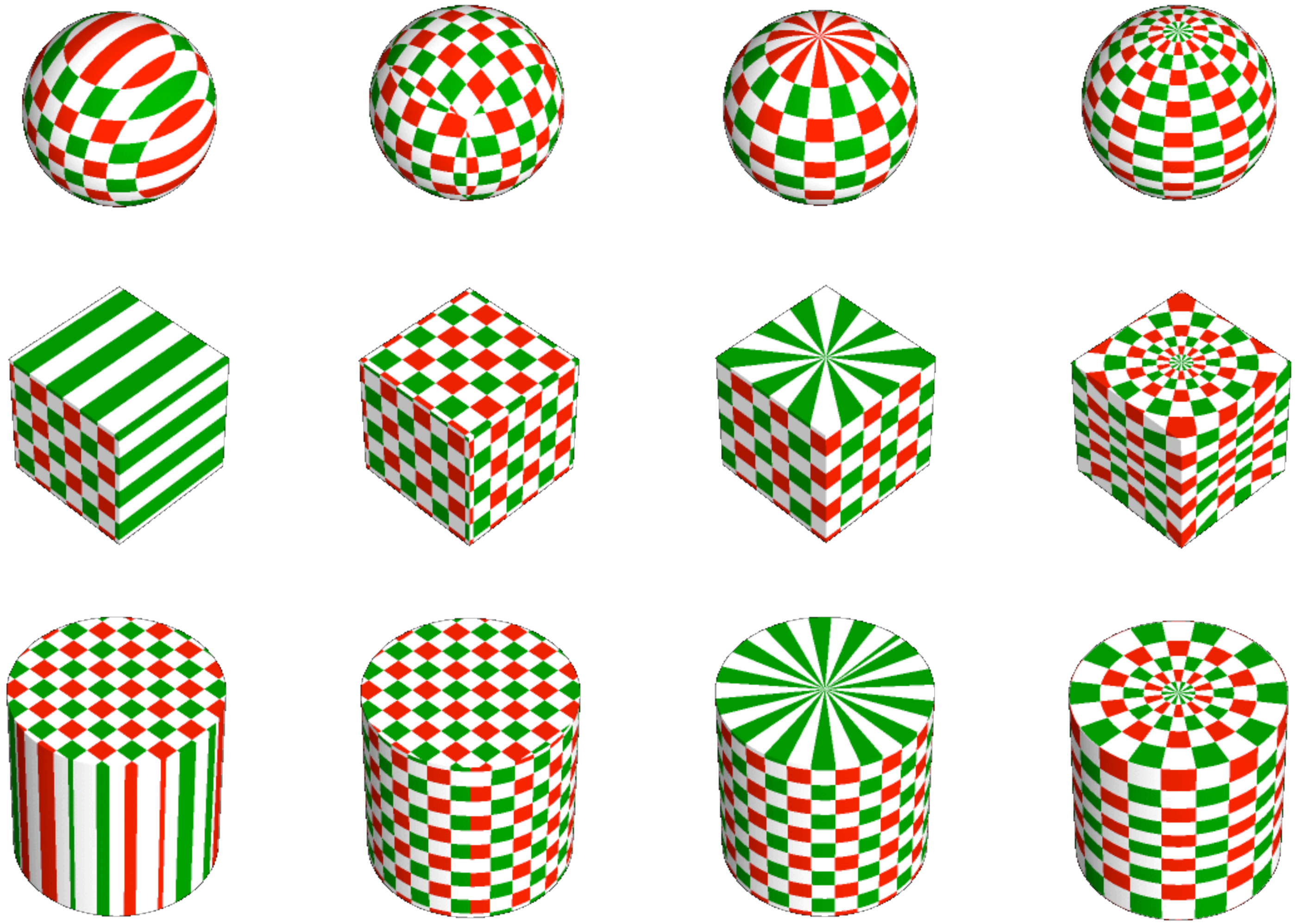
$$r = \sqrt{x^2 + y^2}$$

$$\theta = \tan^{-1}\left(\frac{y}{x}\right)$$

$$z = z,$$



Intermediate Mapping



[Paul Bourke]

Solid Texturing

- Sculpt your object out of a 3D texture
  - Texture is a block or *texture volume* of color values (or other attributes)
  - Immerse the object in the block
  - Each point on the object is assigned the texture attribute from the texture volume

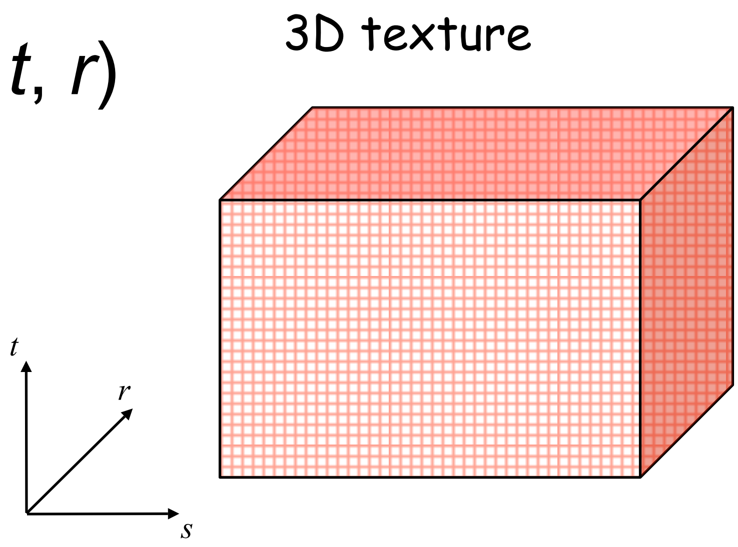
Solid Texturing Effects

- Wood, marble, noisy/bumpy objects

PLATE 4 *Use of shaders wood() (Listing 16.15) on the near elephant, dented() (Listing 16.16) on the middle, blue_marble() (Listing 16.19) on the far elephant, granite() (Listing 16.18) on the pedestal, and the shadowed spotlight shadowspot() (Listing 16.33) as two of the light sources*

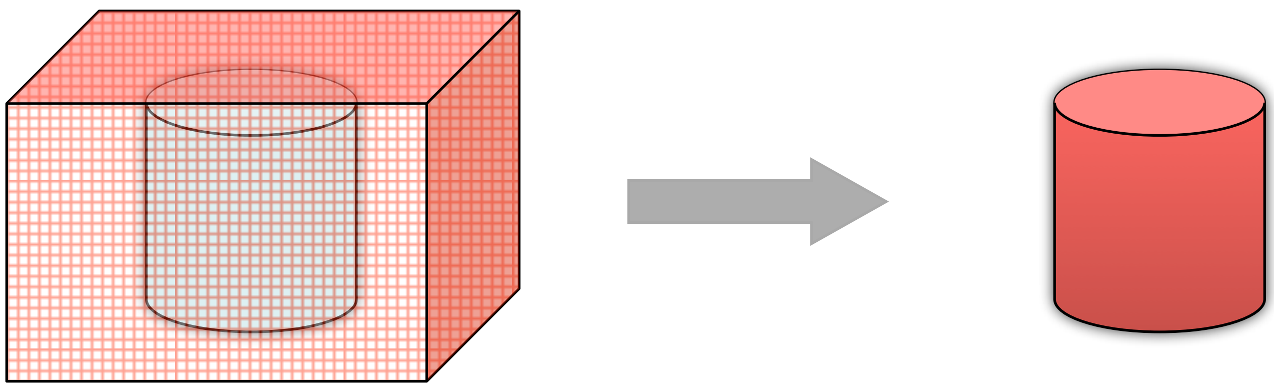
Solid Texturing

- **3D Texture**
  - A 3D array of texel values
  - Texture attributes: Color, ambient, diffuse, specular, opacity
  - Texture space (*s*, *t*, *r*)



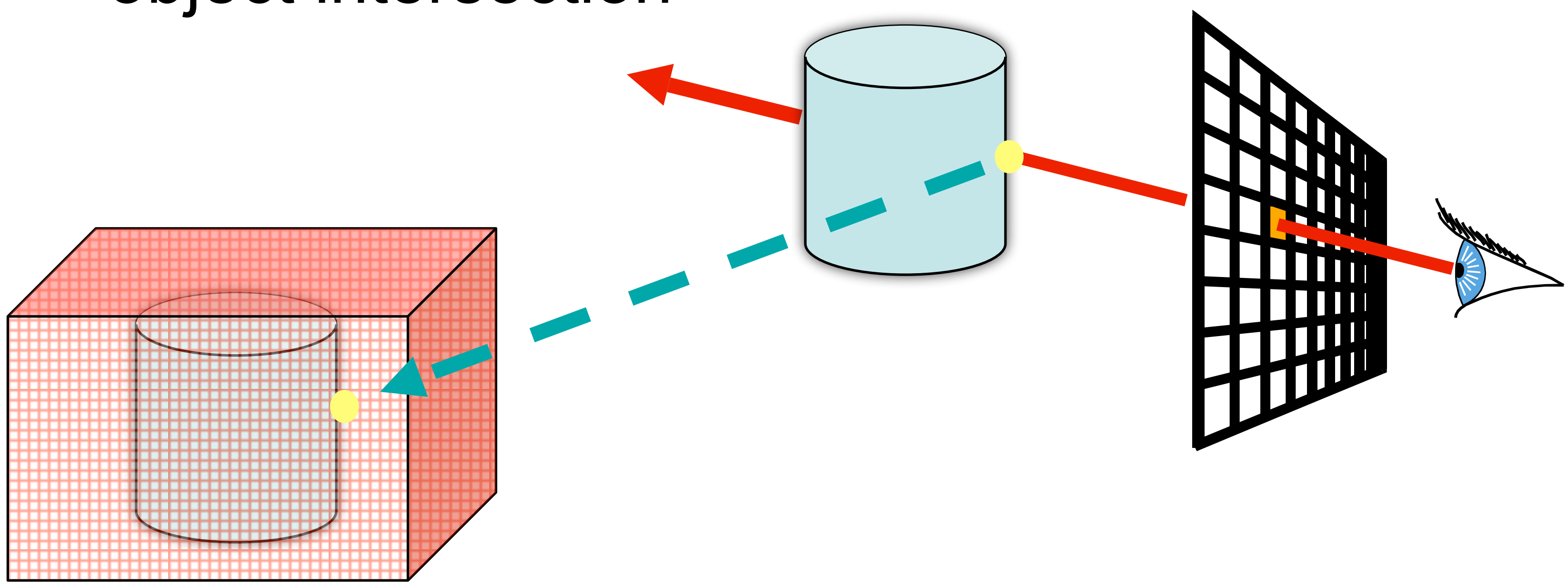
Texture Mapping

- Texture Mapping:
  - Define your object in the texture volume
  - Every object point (x, y, z) –> (s, t, r)



Texture Mapping

- Raytracing a solid textured object
- For each pixel
  - Lookup the texture attribute for each ray-object intersection

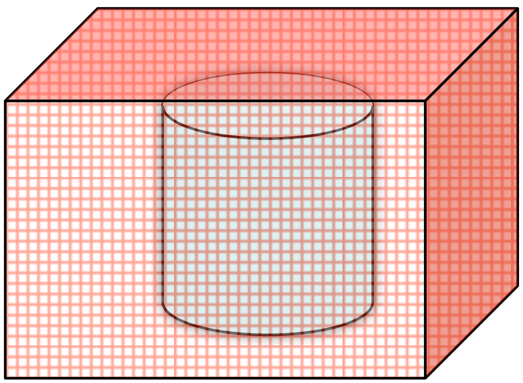


Texture Mapping

- Which space should we define our mapping (x, y, z) –> (s, t, r)?
  - Object or World space coordinates?

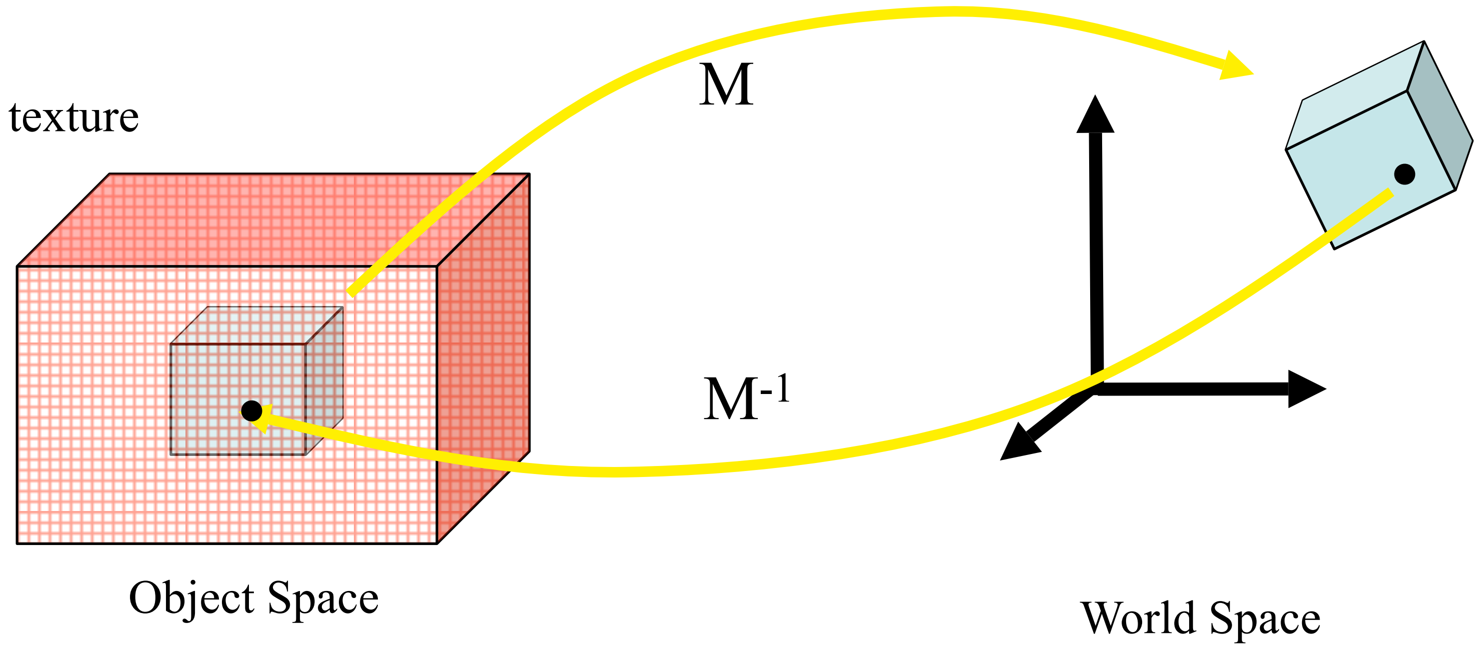
- World Space
  - Static scenes: OK
  - Animated scenes: Object moves through texture

- Object space
  - Texture is ‘fixed’ to object
  - Inverse transform intersection
    - Or trace inverse ray in object space



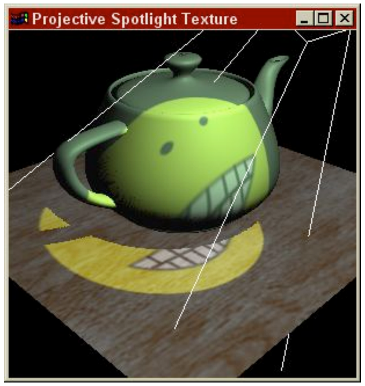
Texture Mapping

- Texture coordinates defined in object space
- (x_w, y_w, z_w) −> (x_o, y_o, z_o) −> (s, t, r)

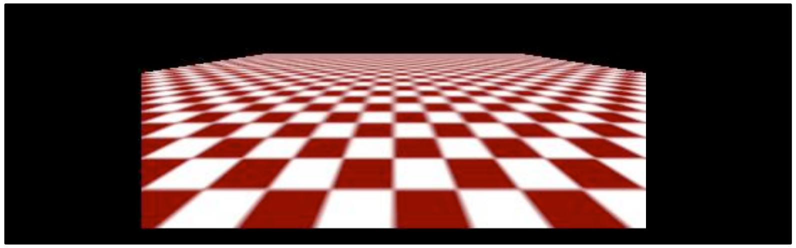


Texture Generation

- Acquiring a 2D texture
  - Scanned photograph
  - Artistic drawing



- How do you acquire a 3D texture?-
  - Procedural textures?

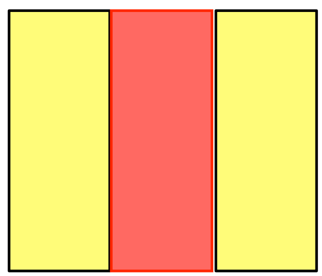


Space Filling Stripes

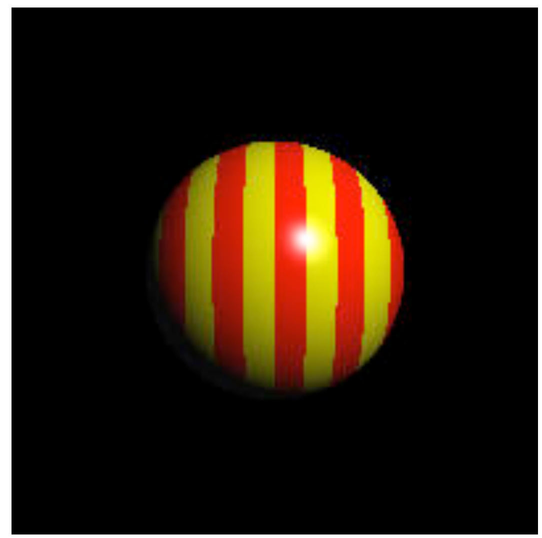
- Computational tool: Modulo divisor %
- Example: Stripes in the *x*-direction

```
rgb Stripes(x, y, z)
{
     jump = ((int)(x)) % 2
     if (jump == 0)
               return yellow
     else
               retiurn red
}
```

0…..1…..0



0...s.x...2*s.x..3*s.x



```
jump = ((int)(A + x/s.x) % 2
if (jump == 0)
     return yellow
```

Strips using the sine function

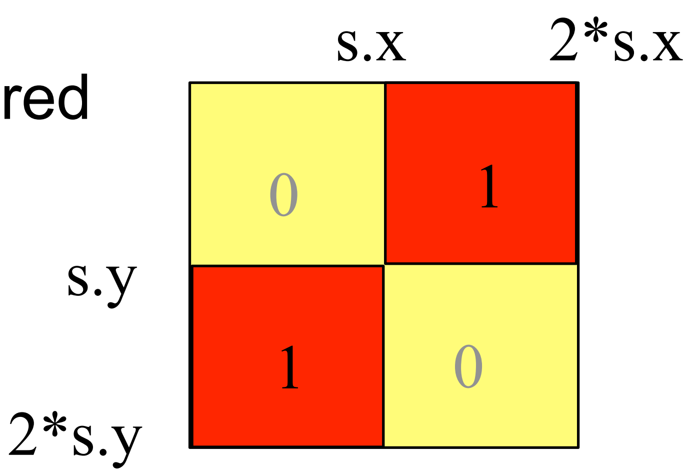
```
Color  stripe(point p)
    if (sin(p.x) > 0)  then
        return c0
    else
        return c1
```

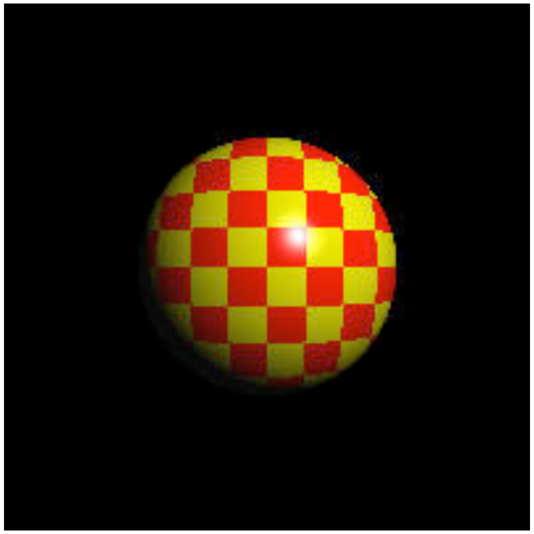
```
//control the width
Color  stripe(point p, width )
    if (sin(PI * p.x/width) > 0) then
        return c0
    else
        return c1
```

1. You can change to p.y or p.z to calculate the strips
2. Question: how do you smoothly transition between c0 and c1?

Space Filling 2D Checkerboard

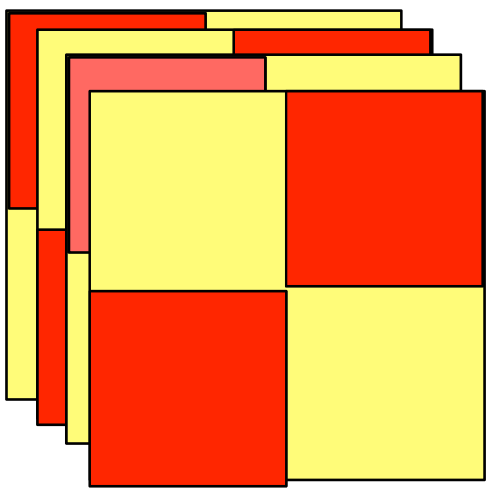
```
rgb 2DCheckerboard(x, y, z)
{
    jump = ((int)(A + x/s.x) + (int)(A + y/s.y)) % 2
    if (jump == 0)
        return yellow
    else
        return red
}
```

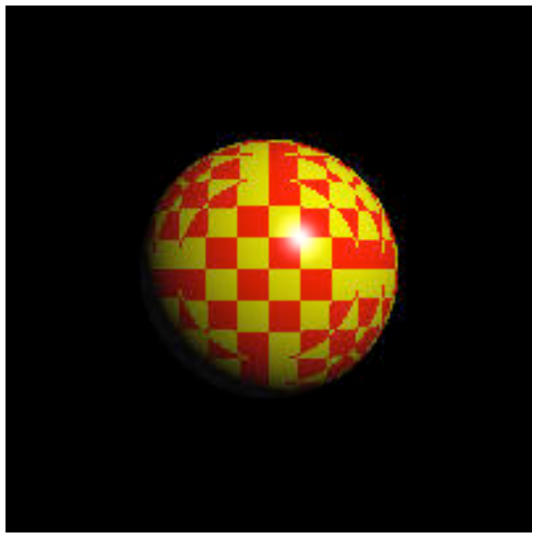




Space Filling 3D Checkerboard

```
rgb 3DCheckboard(x, y, z)
{
    jump = ((int)(A + x/s.x)+(int)(A + y/s.y) )+(int)(A+z/
    s.z))%2
    if (jump == 0)
        return yellow
    else
        return red
}
```





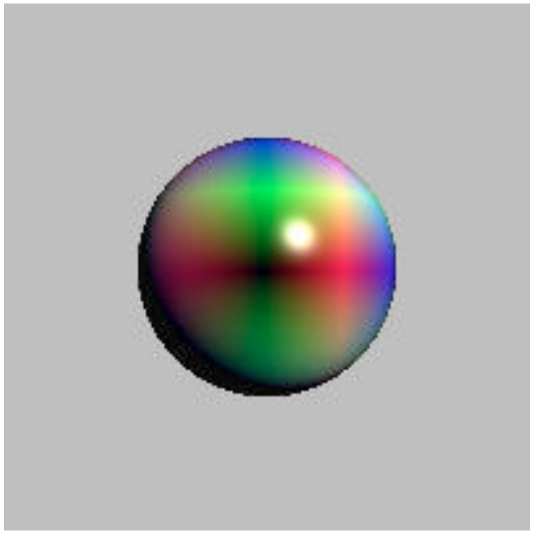
Cube of Smoothly Varying Colors

- Computational tool: floor or ceil

  Let fract(x) = x - *floor*(x)

  (r, g, b) = (1 - |2*fract(x) - 1|, 1-|2*fract(y) - 1|,
  1-|2*fract(z) - 1|)

0….1…..0

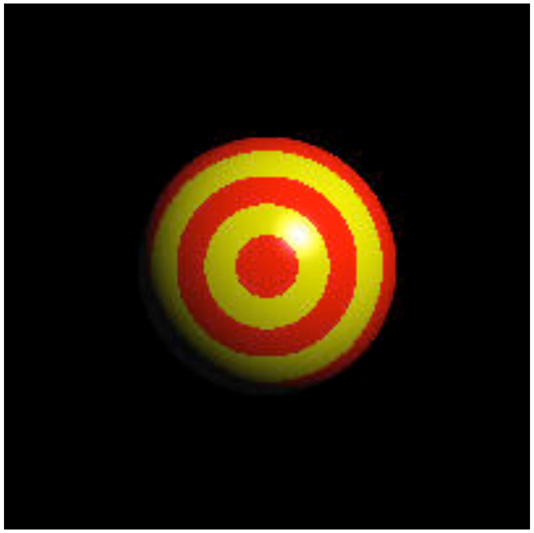


Rings

- Concentric Circles

Let rings(*r*) = (int (*r*)) % 2, where $r = \mathrm{sqrt}(x^2 + y^2)$;

rings(*r*) = D + A * rings(r/M)
where M - thickness

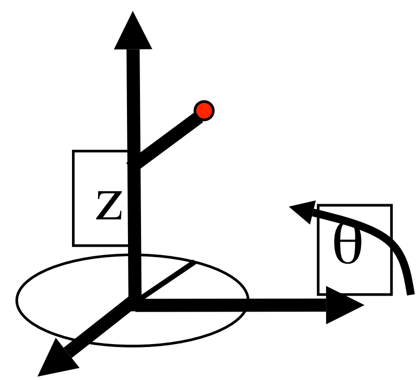


Wood Grain

Wobble: $\text{rings}(r) = \text{rings}(r/M + k*\sin(\theta/N))$

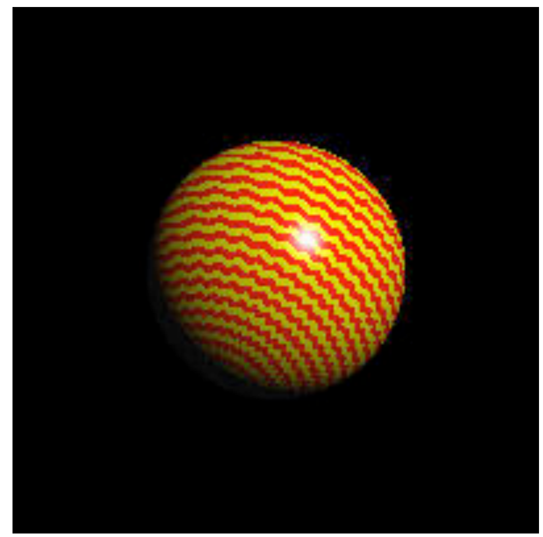
Twist: $\text{rings}(r) = \text{rings}(r/M + k*\sin(\theta/N + Bz))$

θ – Azimuth around the z-axis

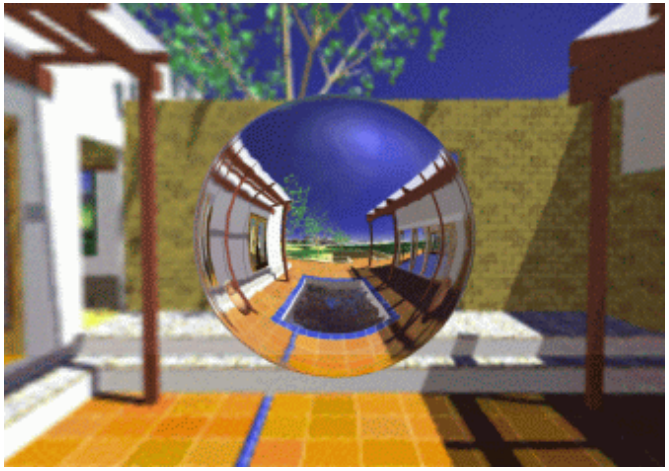


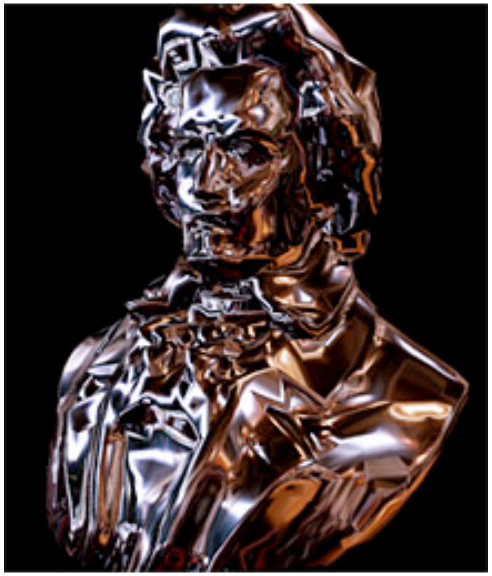
Wood Grain

To tilt the grain,
$(x',y',z') = T(x,y,z)$
for some rotational transform T



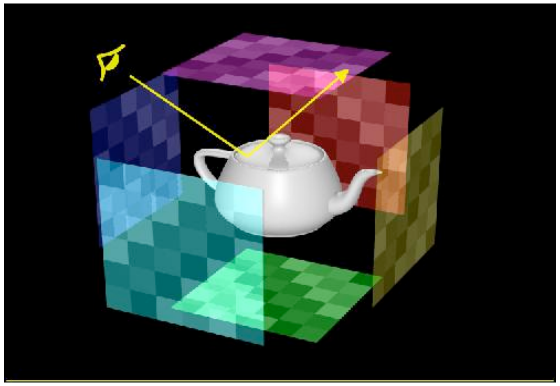
Environment Mapping

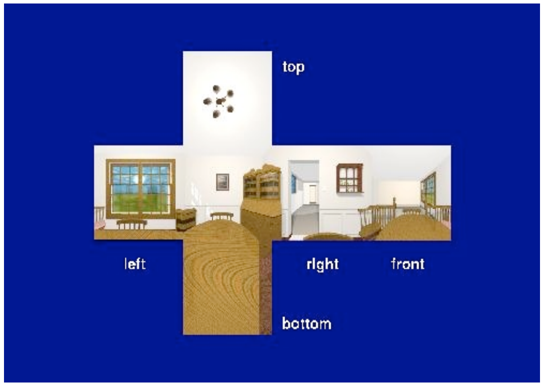




Environment Mapping

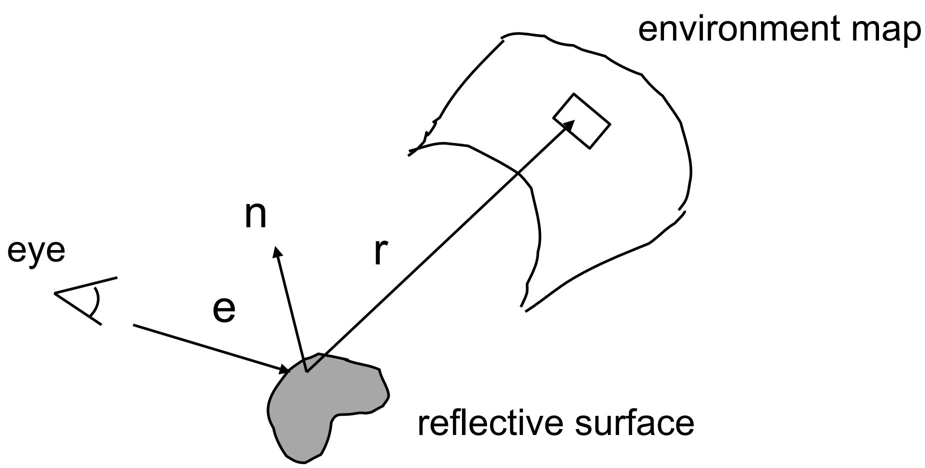
- Also called reflection mapping
- First proposed by Blinn and Newell 1976
- A cheap way to create reflections on curved surfaces – can be implemented using texture mapping supported by graphics hardware

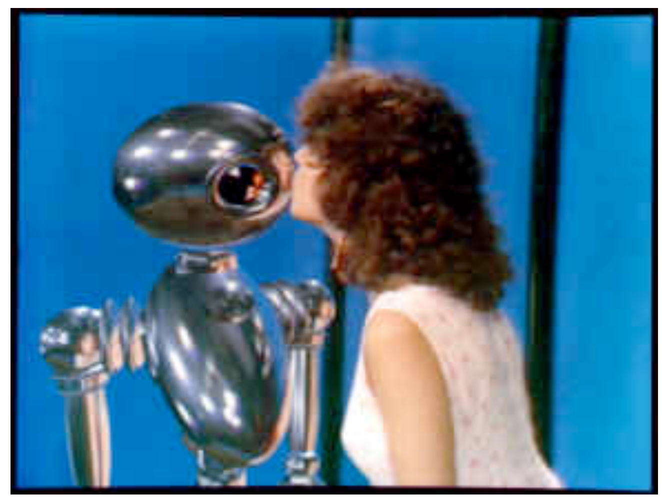




Basic Idea

- Assuming the environment is far away and the object does not reflect itself – the reflection at a point can be solely decided by the reflection vector



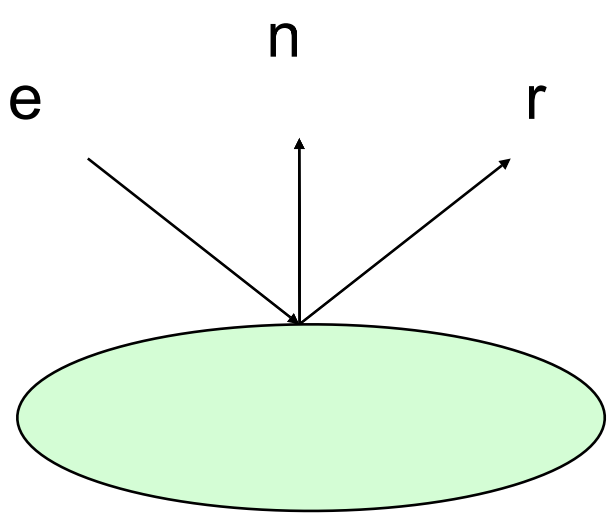


Basic Steps

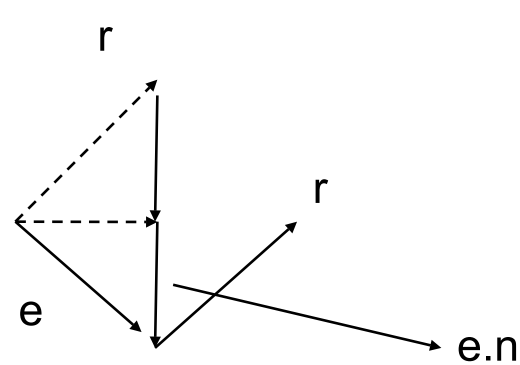
- Create a 2D environment map
- For each pixel on a reflective object, compute the normal
- Compute the reflection vector based on the eye position and surface normal
- Use the reflection vector to compute an index into the environment texture
- Use the corresponding texel to color the pixel

Finding the reflection vector

- r = e – 2 (n.e) n

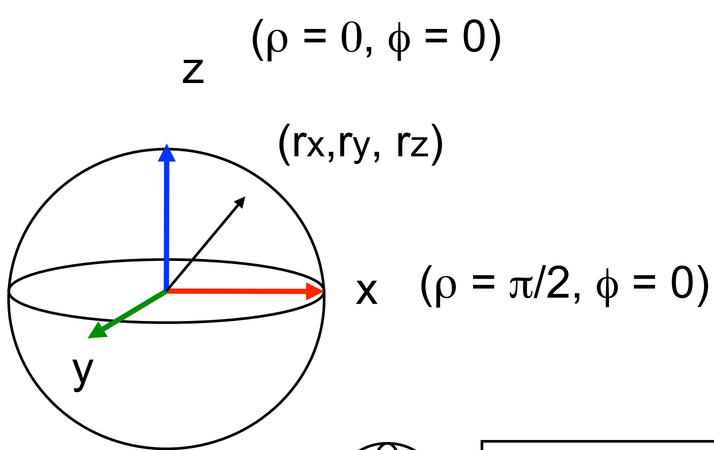


Assuming e and n are all normalized

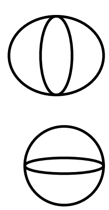


Blinn and Newell's

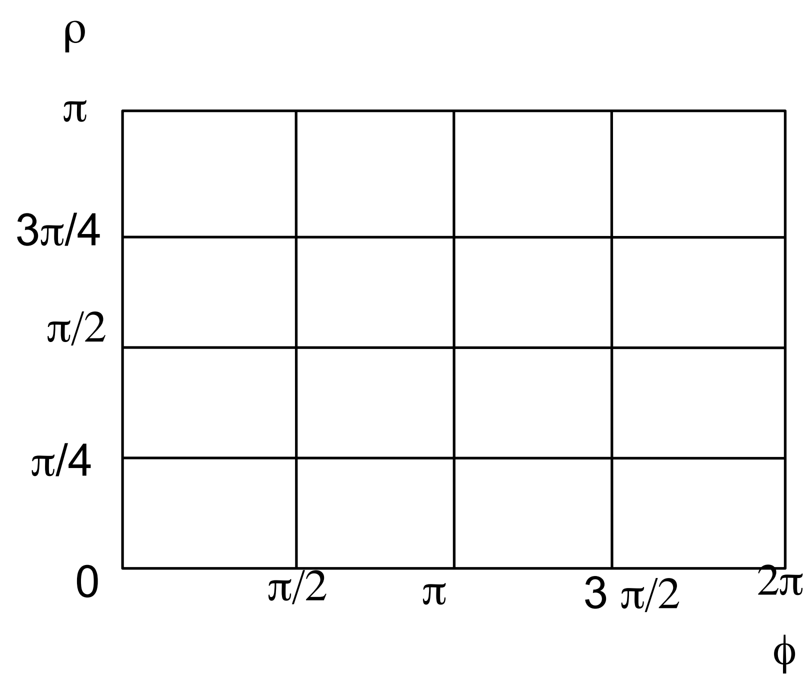
- Blinn and Newell's Method (the first EM algorithm)
- Convert the reflection vector into spherical coordinates (ρ,ϕ), which in turn will be normalized to [0,1] and used as (u,v) texture coordinates



ρ: latitude $[0, \pi]$
ϕ: longitude $[0, 2\pi]$

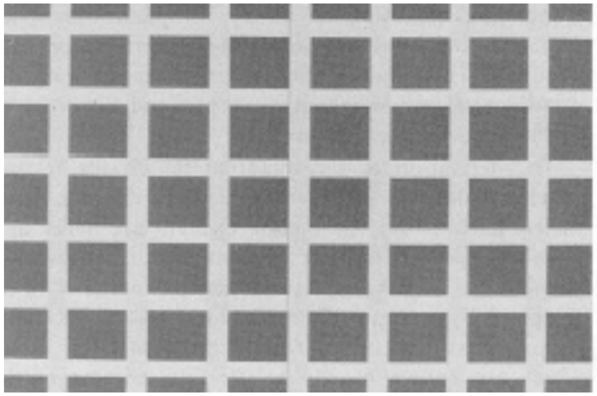


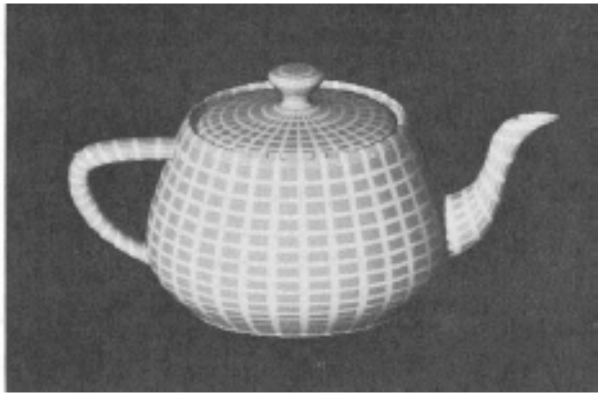
ρ = arccos(-rz)
ϕ = atan2(ry, rx)



Issues

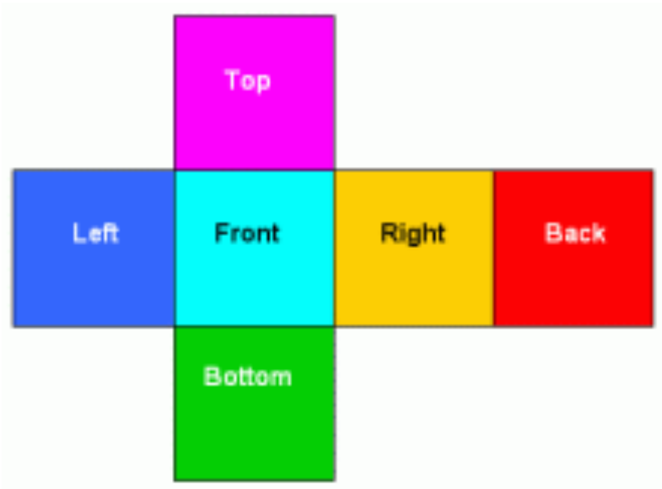
- Seams at $\phi = 0$ when the triangle vertices span over
- Distortion at the poles, and when the triangle vertices span over
- Not really been used much in practice

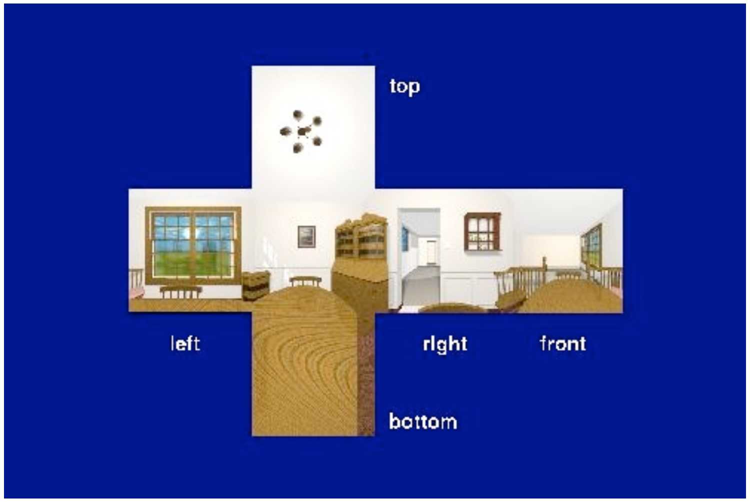




Cubic Environment Mapping

- Introduced by Nate Green 1986 (also known as environment cube map)
- Place the camera in the center of the environment and project it to 6 sides of a cube





Cubic Environment Mapping (2)

- Texture mapping process
  - Given the reflection vector (x,y,z), first find the major component and get the corresponding plane. (-3.2, 5.1, -8.4) -> -z plane
  - Then use the remaining two components to access the texture from that plane.
    - Normalize them to (0,1)

      (-3.2, 5.1) -> (-3.2/8.4)+1, 5.1/8.4+1)
    - Then perform the texture lookup
- No distortion or seam problems, although when two vertices of the same polygon pointing to different planes need to be taken care of.

Environment Cube Map

- Rendering Examples