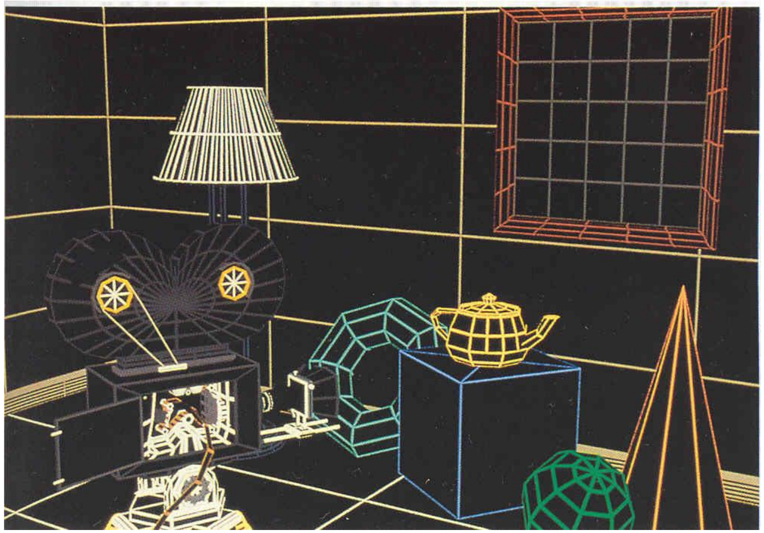
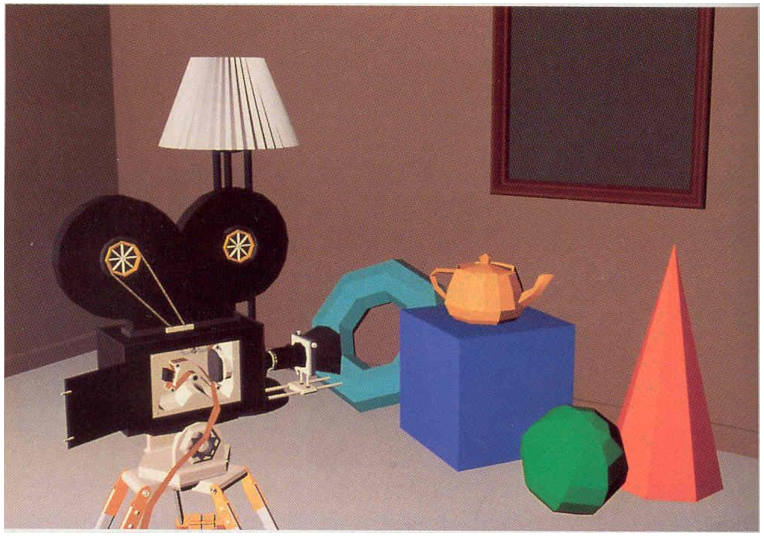
Texture Mapping

Fall 2023
11/16/2023
Kyoung Shin Park
Computer Engineering
Dankook University

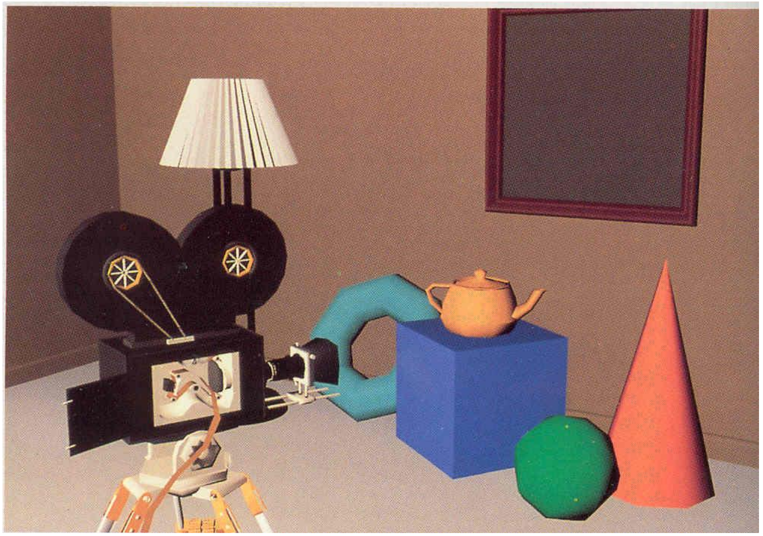
Texture Mapping



Wireframe



Flat shading



Smooth shading

Texture mapping

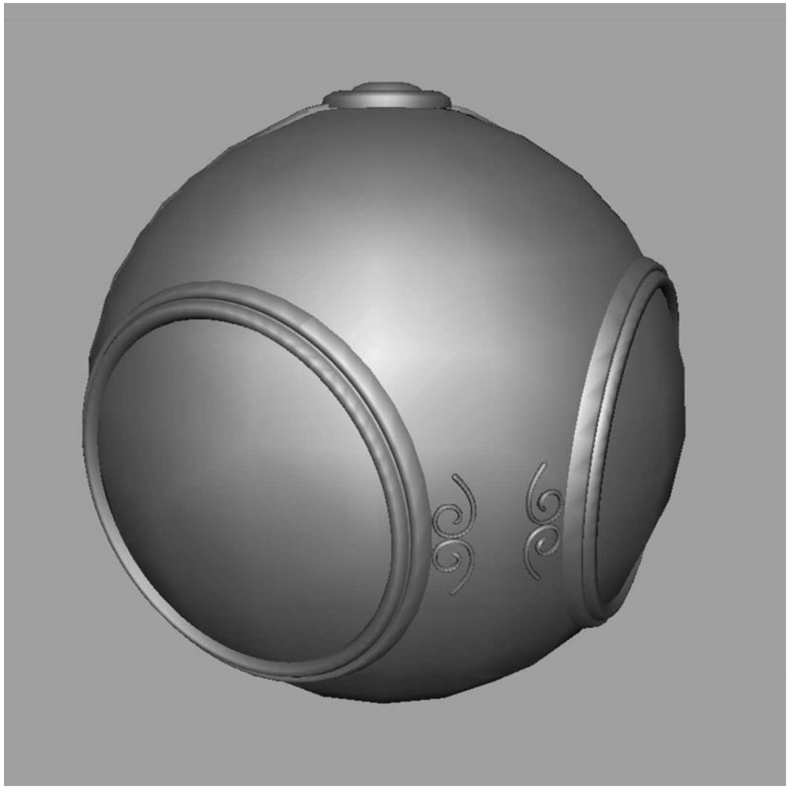
The Limits of Geometric Modeling

- Although graphics cards can render over 10 million polygons per second, that number is insufficient for many phenomena
  - Clouds
  - Grass
  - Terrain
  - Skin
- Texture Mapping
  - Two-dimensional image is applied directly to a surface
  - In real-time graphics rendering where a limited number of polygons must be used, texture mapping is a technique that can significantly increase the realism with a relatively small additional cost.

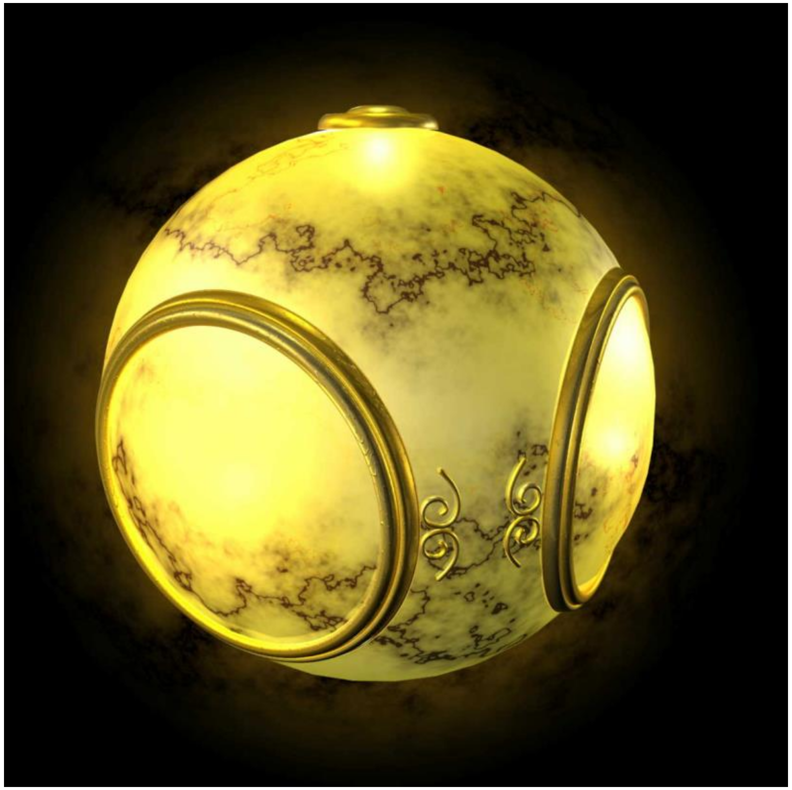
Three Types of Mapping

- Texture Mapping
  - Uses images to fill inside of polygons.
- Environment/Reflection mapping
  - Uses a picture of environment for texture maps.
  - Allows simulation of highly specular surfaces.
- Bump mapping
  - Emulates altering normal vectors during the rendering process.

Texture Mapping



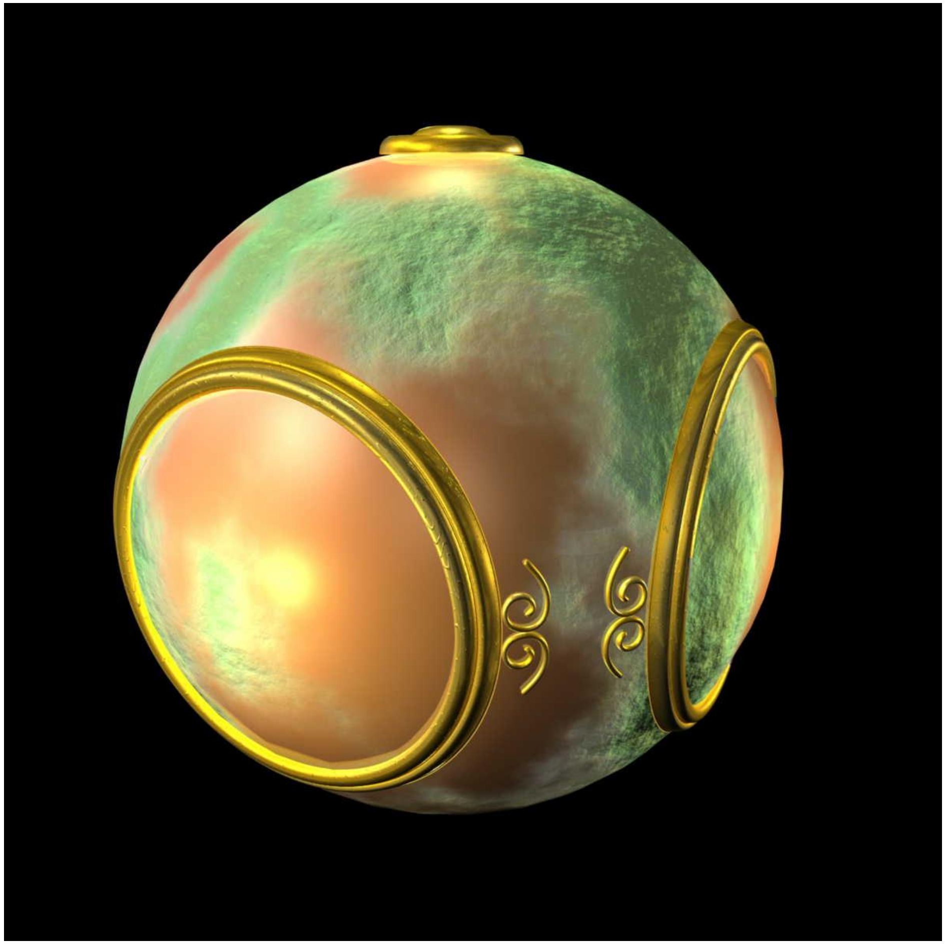
geometric model



texture mapped

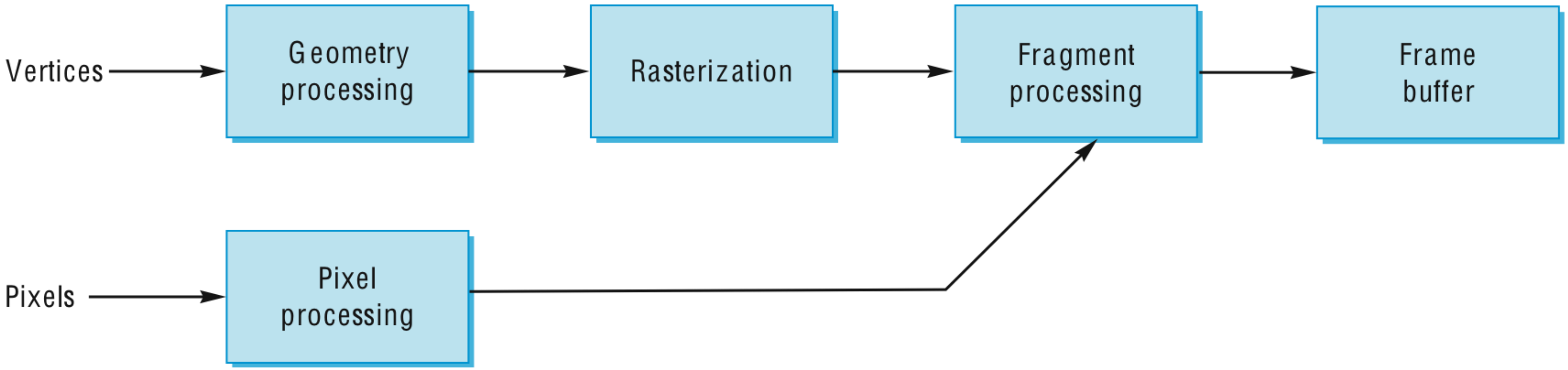
Environment Mapping

Bump Mapping



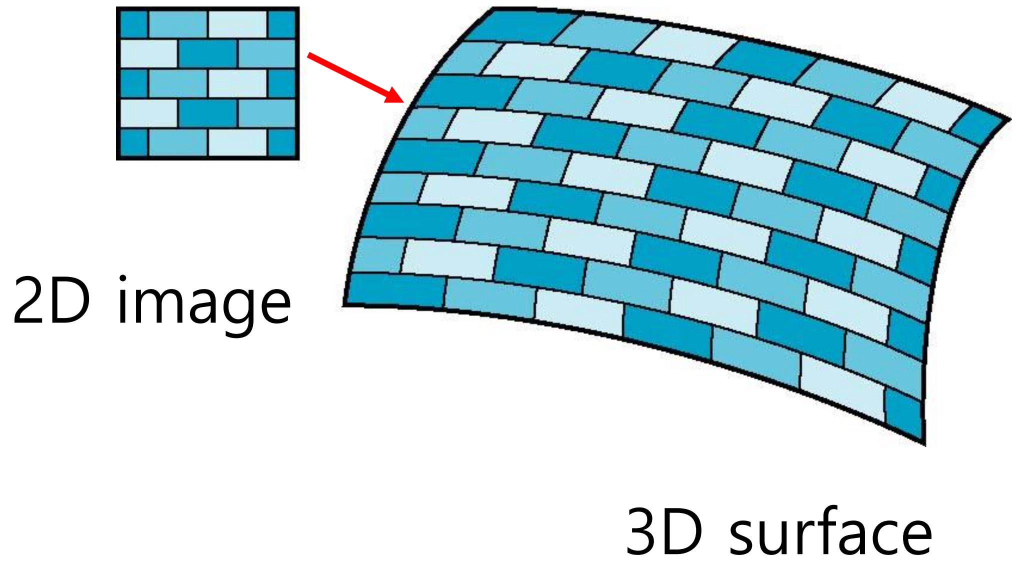
Where does texture mapping take place?

- Mapping techniques are implemented at the end of the rendering pipeline
  - Very efficient because few polygons make it past the clipper



Is it simple?

- Although the idea is simple - map an image to a surface.
  - There are 3 or 4 coordinate systems involved



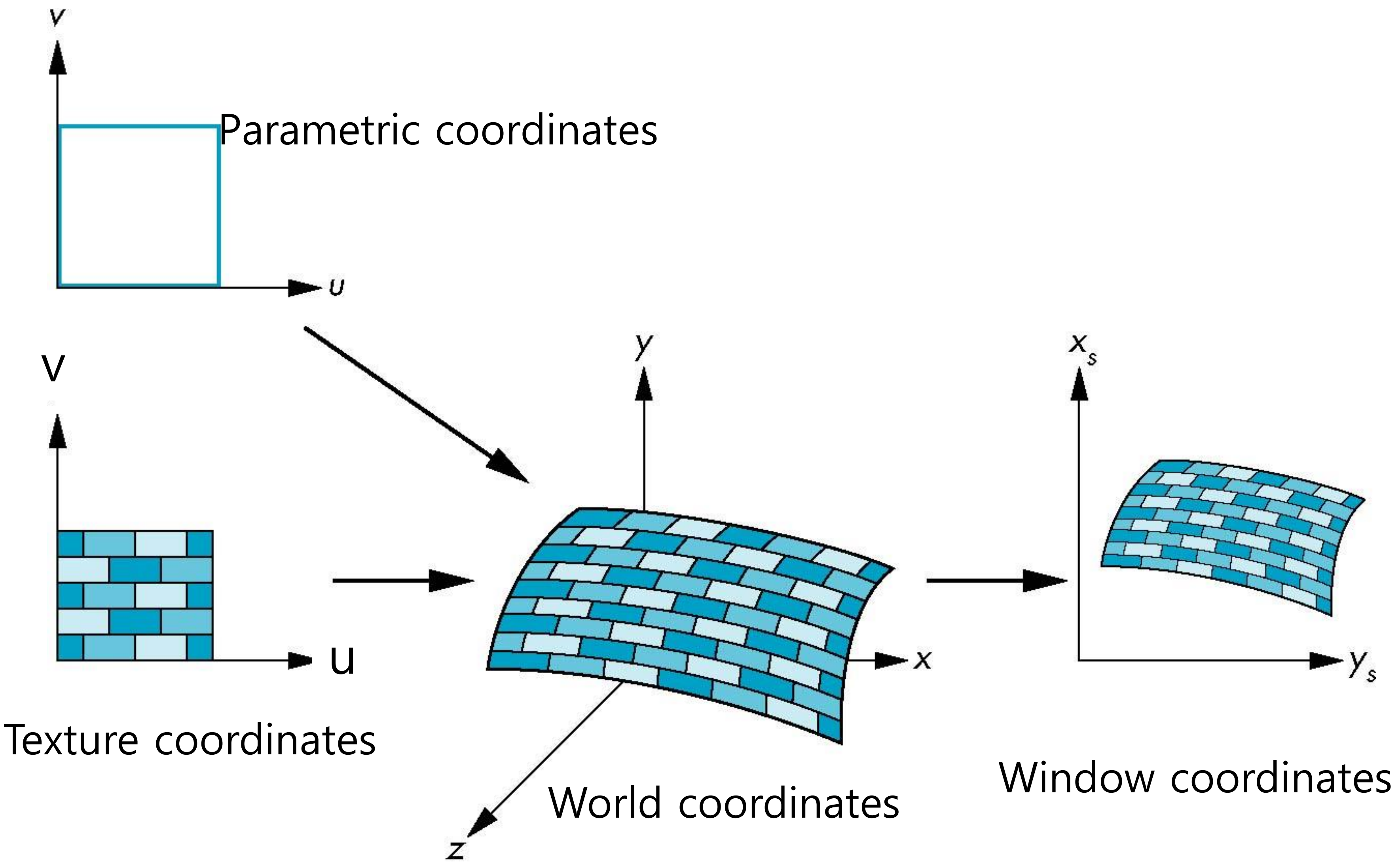
Texture Mapping

- Conceptual 2D texture mapping process
  - Surface parameterization
    - How to apply a texture image to an object?
    - The coordinates of the texture image mapped to each point of the object are required. $(x_0, y_0, z_0) => (x_t, y_t)$
  - Geometric transformation
    - Geometric transformation determines the mapping relationship between each point of an object and its position on the projection screen. $(x_0, y_0, z_0) => (x_s, y_s)$
  - Rasterization
    - The process of finding pixels on which each geometric object is projected
  - Texture color calculation
    - The process of painting each pixel with a texture color appropriately
    - How to calculate the texture color visible through each pixel?
    - How to blend the calculated texture color with the original color of the object?

Coordinate Systems

- Parametric coordinates (u, v)
  - May be used to model curves & surfaces
- Texture coordinates (u, v)
  - Used to identify points in the image to be mapped
- Object or World Coordinates (x, y, z)
  - Conceptually, where the mapping takes place
- Window Coordinates (x_s, y_s)
  - Where the final image is really produced

Texture Mapping



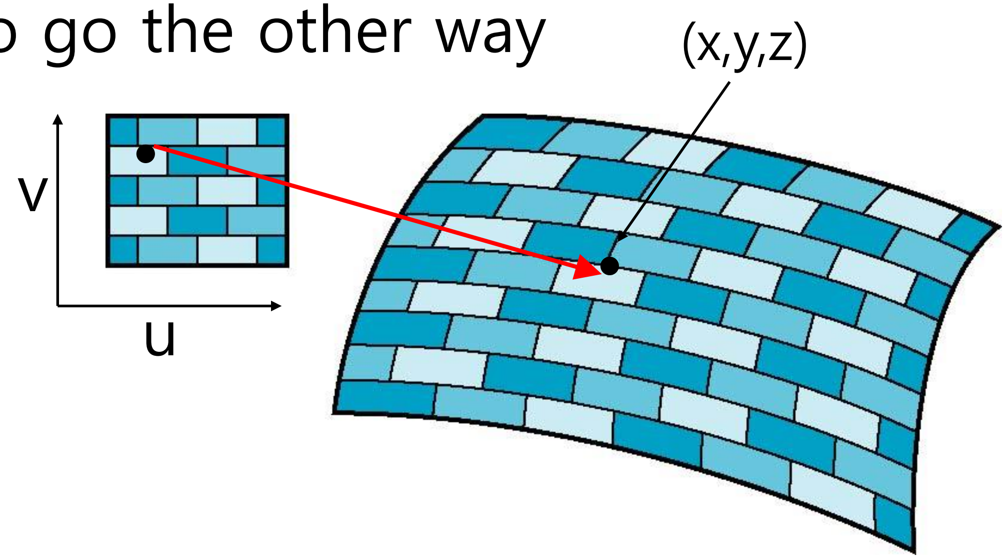
Mapping Functions

- Basic problem is how to find the maps
- Consider mapping from texture coordinates to a point a surface
- Appear to need three functions

  x = x(u,v)

  y = y(u,v)

  z = z(u,v)
- But we really want to go the other way

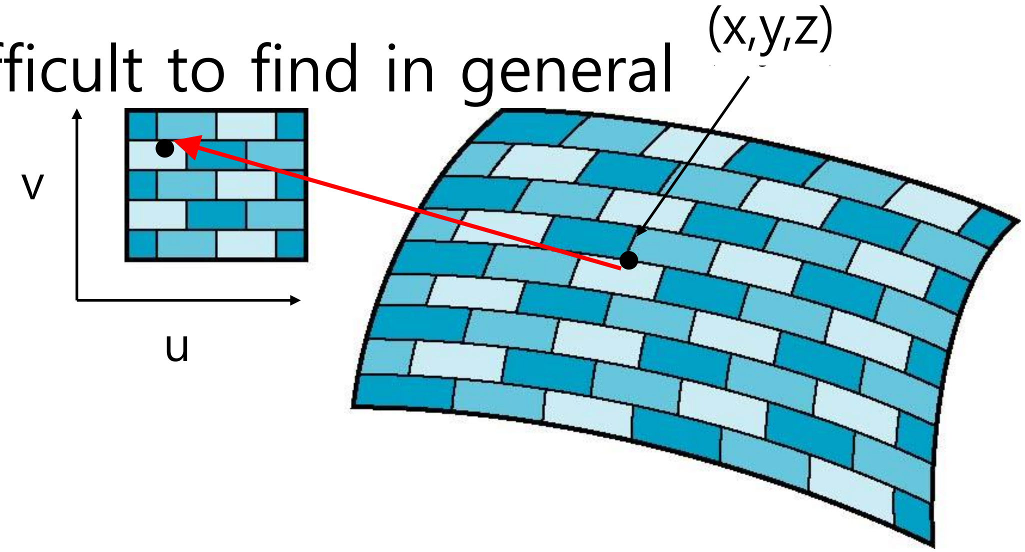


Backward Mapping

- We really want to go backwards
  - Given a pixel, we want to know to which point on an object it corresponds
  - Given a point on an object, we want to know to which point in the texture it corresponds
- Need a map of the form

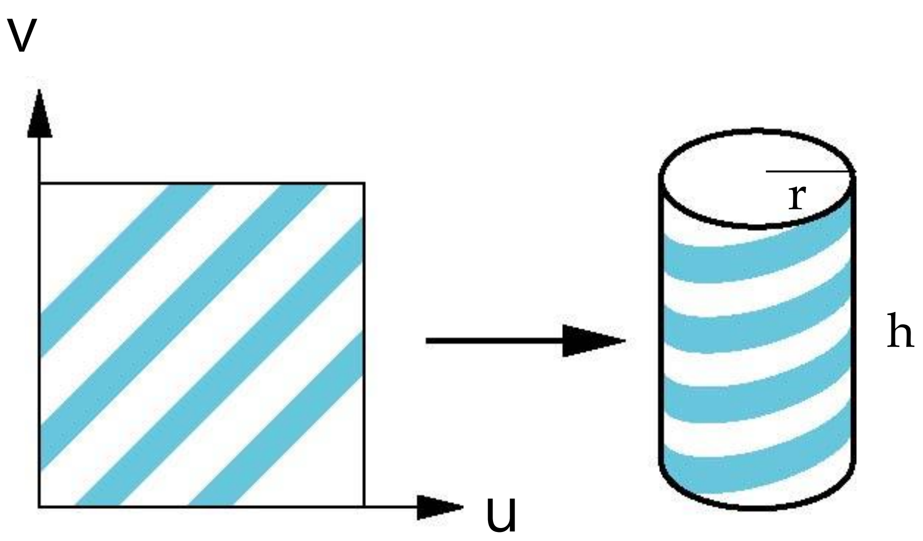
  u = u(x,y,z)

  v = v(x,y,z)
- Such functions are difficult to find in general



Two-part mapping

- Two-part mapping
  - One solution to the mapping problem is to first map the texture to a simple intermediate surface
- Example: first, map to cylinder



Cylindrical Mapping

- Parametric cylinder

  $x = r \cos 2\pi u$ u: (0,1)

  $y = r \sin 2\pi u$ v: (0,1)

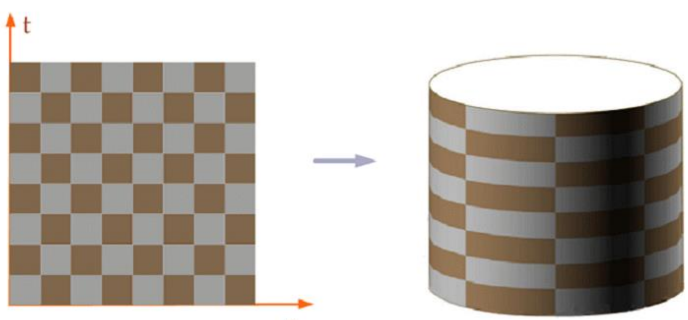
  $z = v/h$

- Maps rectangle in u,v space to cylinder of radius r and height h in world coordinates

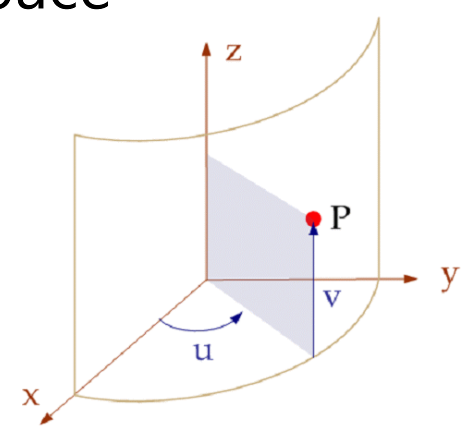
  $u = u$

  $v = v$

- Then, maps from texture space



(a) (b)



$x = r \cos 2\pi u$
$y = r \sin 2\pi u$
$z = v/h$

Spherical Map

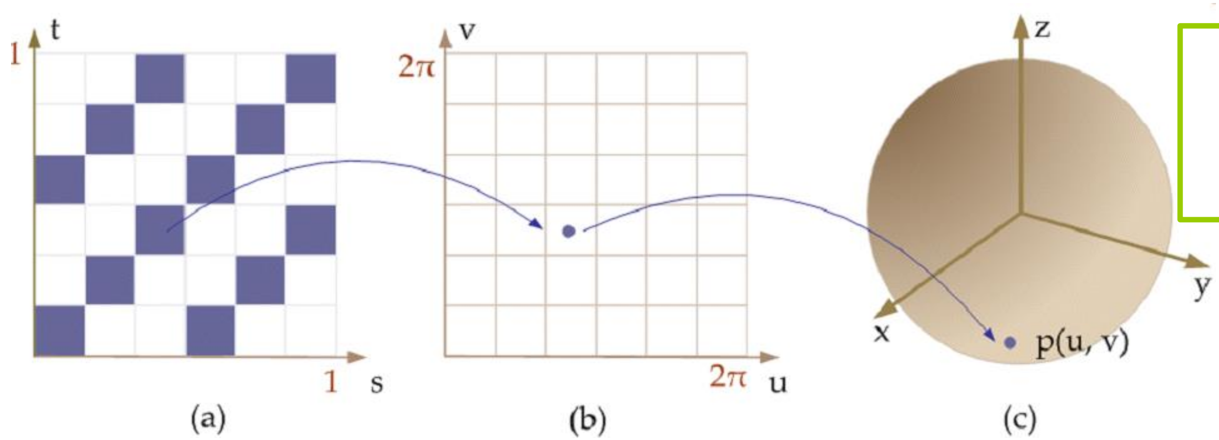
- We can use a parametric sphere

  $x = r \cos 2\pi u$

  $y = r \sin 2\pi u \cos 2\pi v$

  $z = r \sin 2\pi u \sin 2\pi v$

- In a similar manner to the cylinder but have to decide where to put the distortion
  - Mercator projection creates the largest distortion at both poles.
- Spherical mapping is used in environmental maps.



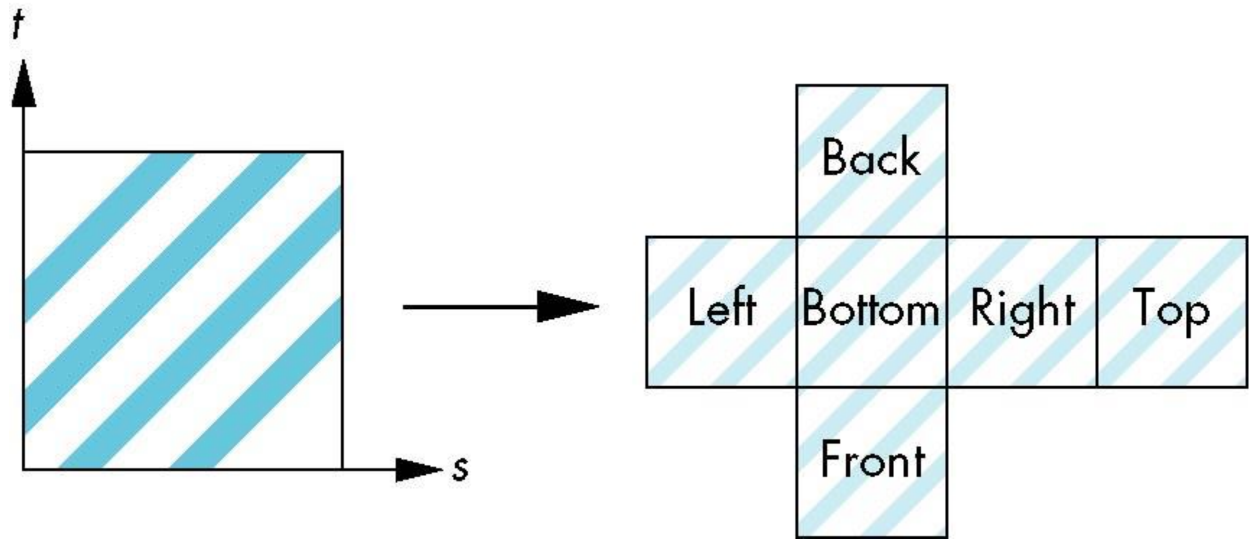
$x = r \cos 2\pi u$

$y = r \sin 2\pi u \cos 2\pi v$

$z = r \sin 2\pi u \sin 2\pi v$

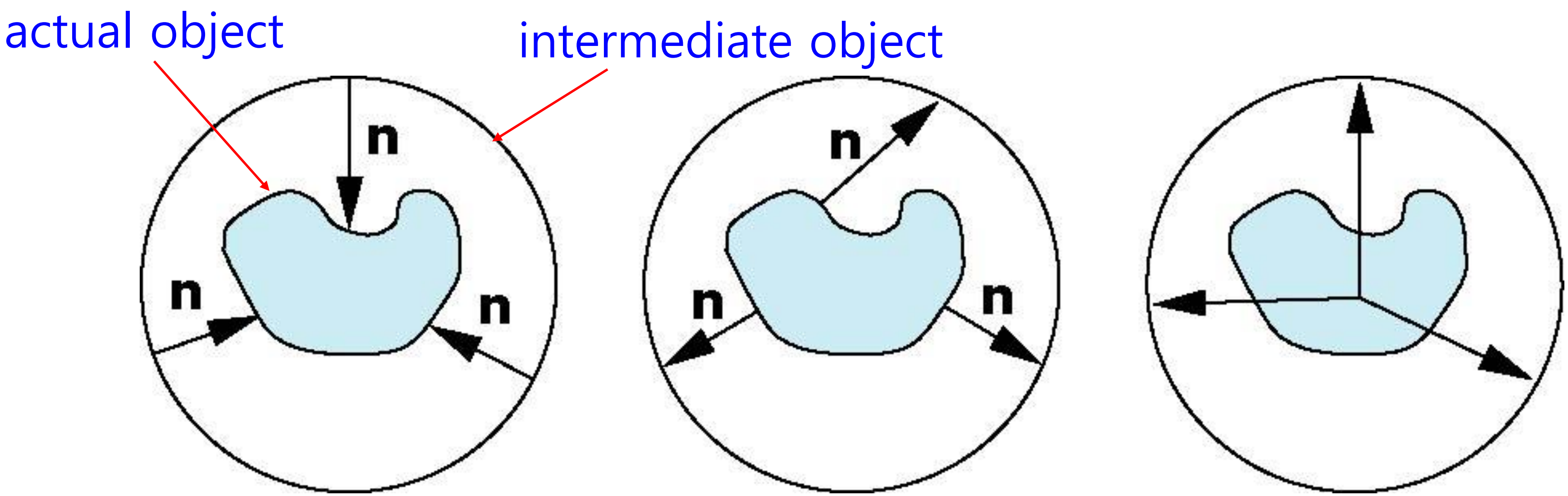
Box Mapping

- Easy to use with simple orthographic projection
- Also used in environment maps



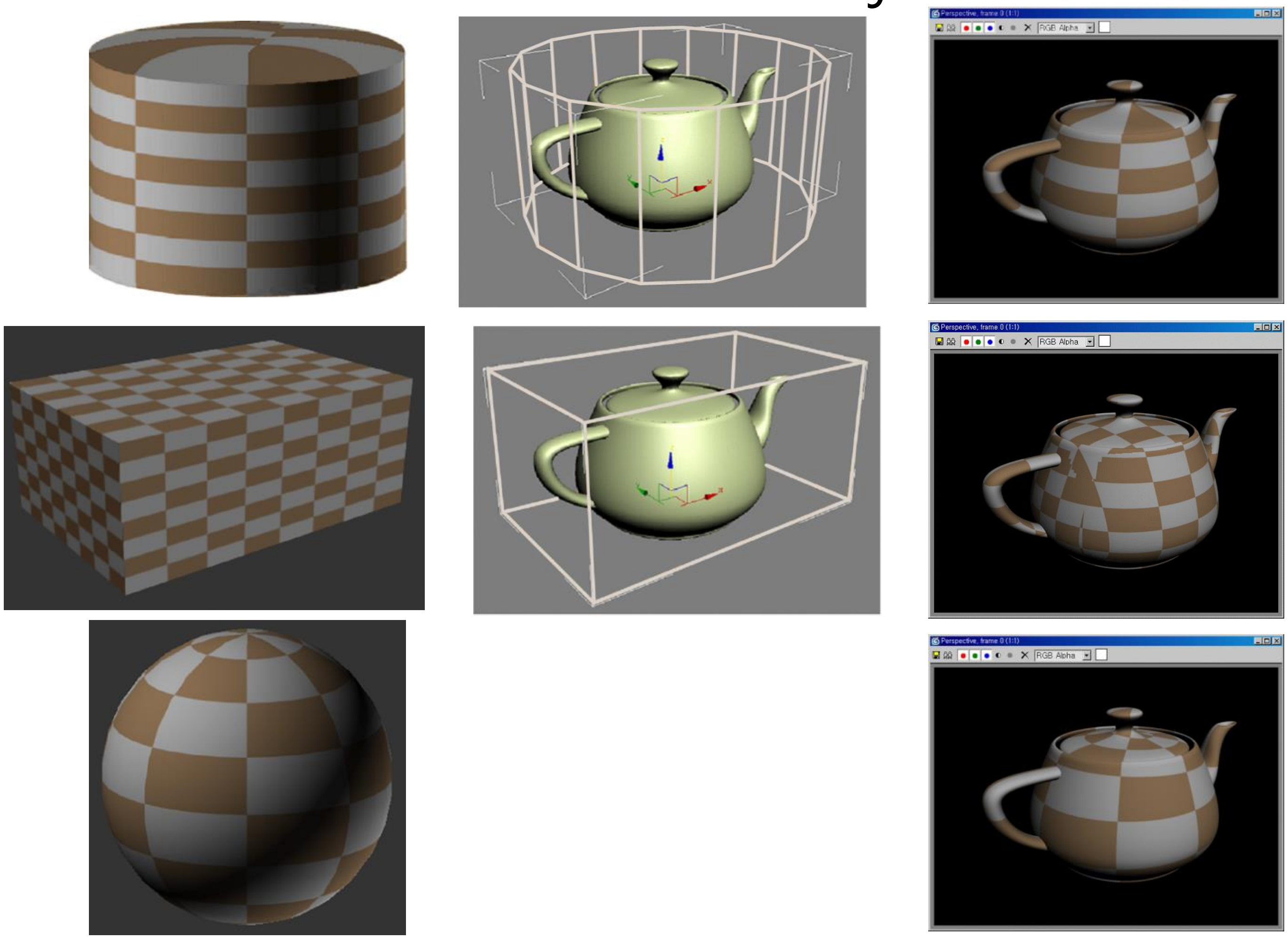
Second Mapping

- Map from intermediate object to actual object
  - Normals from intermediate to actual
  - Normals from actual to intermediate
  - Vectors from center of intermediate



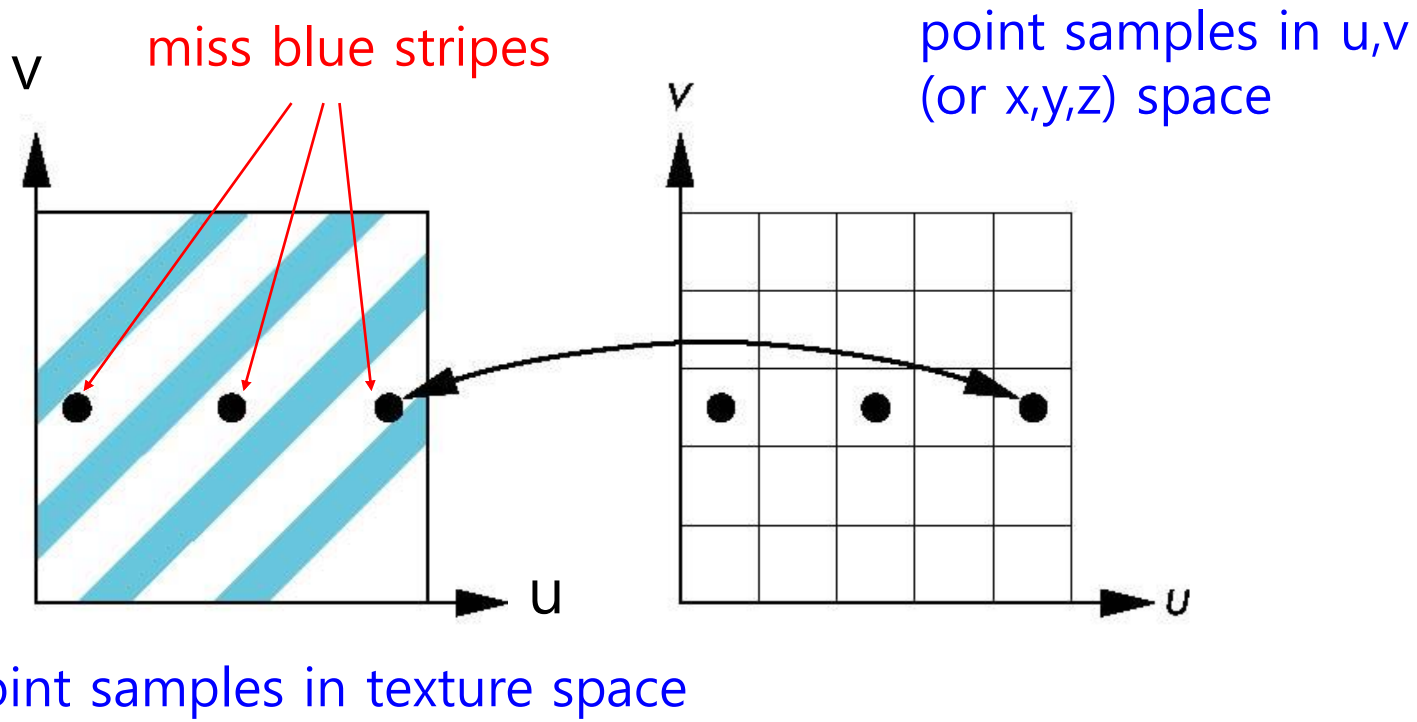
Second Mapping

- Put the object inside the mediation surface and apply texture to the surface of the object.



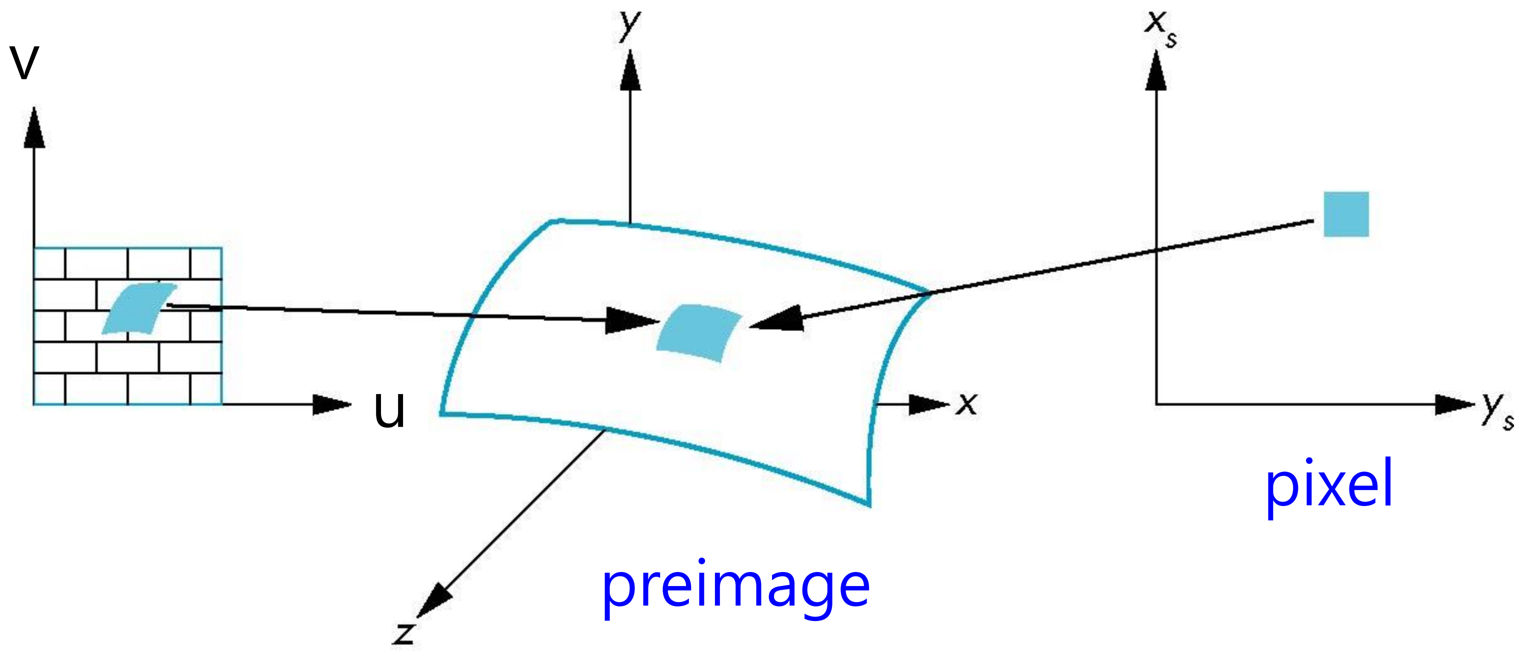
Aliasing

- Point sampling of the texture can lead to aliasing errors
  - Point sampling – point to point mapping



Area Averaging

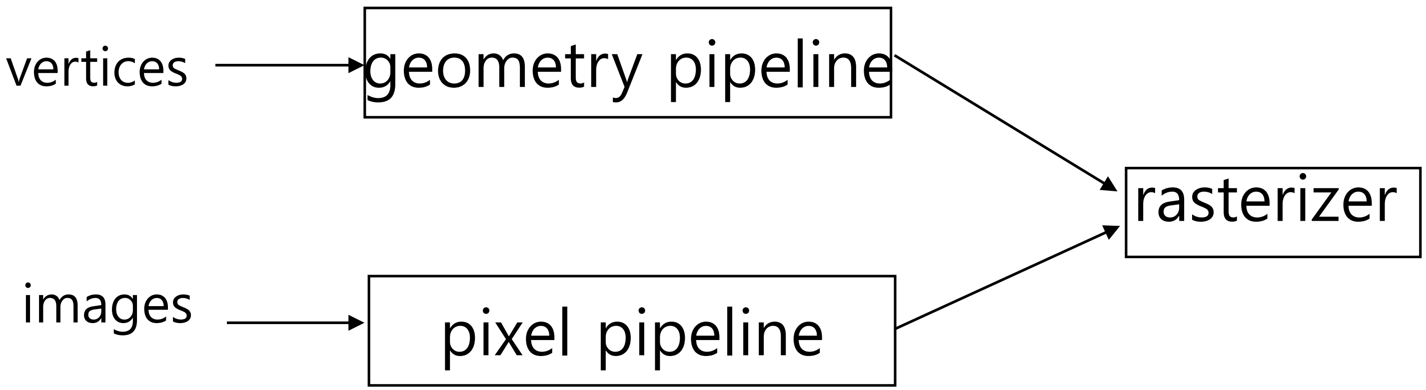
- A better but slower option is to use area averaging
  - Area Averaging – area to area mapping



  - Note: the *preimage* of pixel is curved

Texture Mapping in the Rendering Pipeline

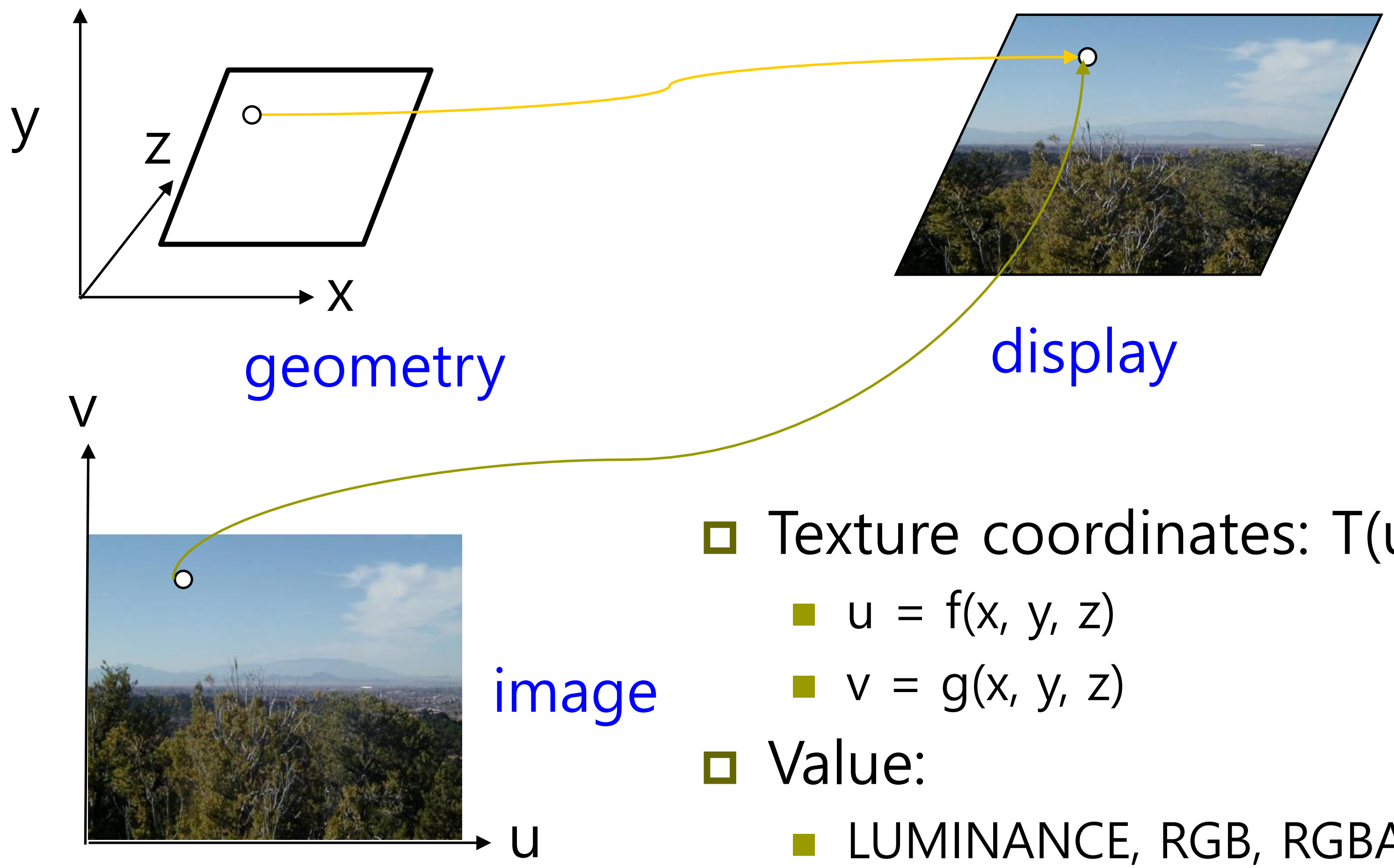
- Images and geometry flow through separate pipelines that join at the rasterizer
  - "complex" textures do not affect geometric complexity



Basic Strategy

- Three steps to applying a texture
  1. Specify the texture
     - Read or generate image
     - Assign to texture
     - Enable texturing
  2. Assign texture coordinates to vertices
     - Proper mapping function is left to application
  3. Specify texture parameters
     - Wrapping
     - Filtering

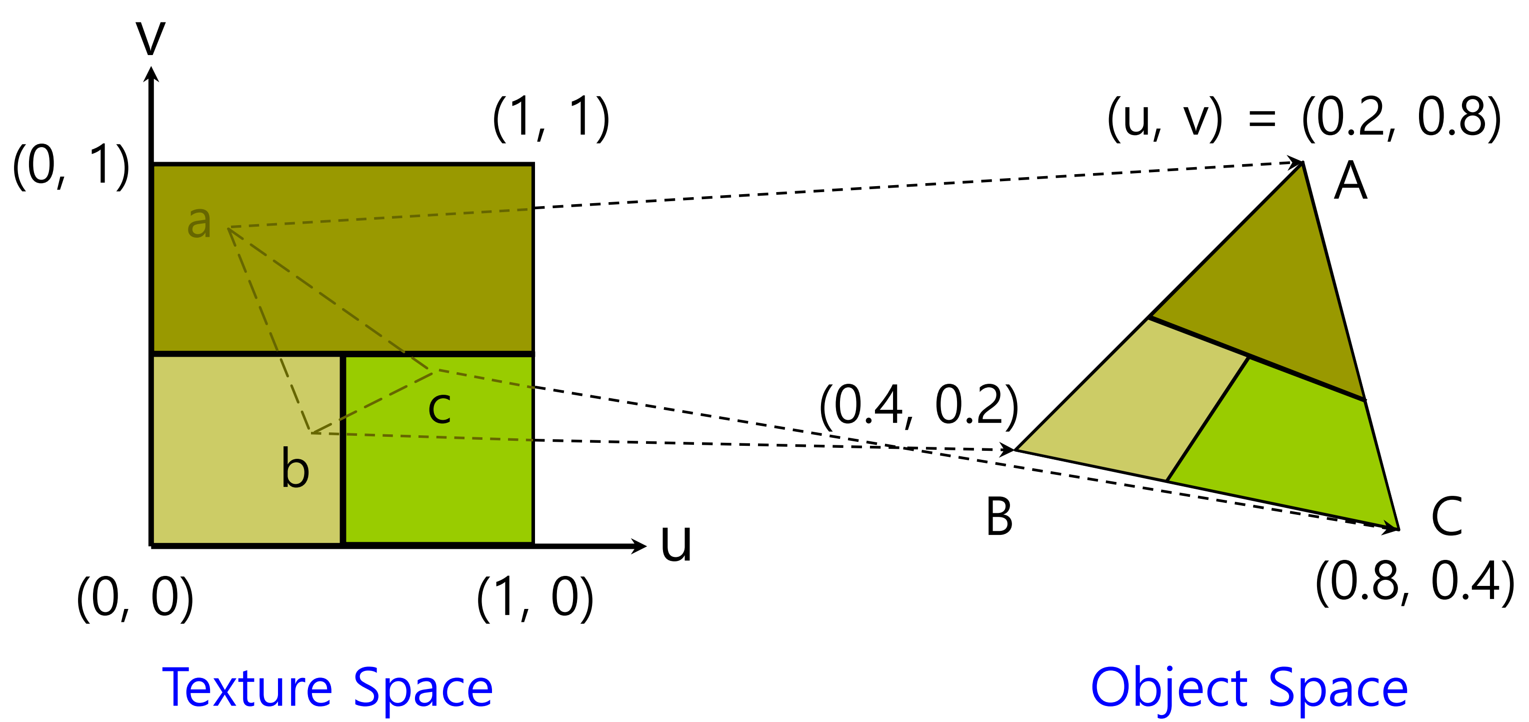
Texture Mapping



- Texture coordinates: T(u, v)
  - u = f(x, y, z)
  - v = g(x, y, z)
- Value:
  - LUMINANCE, RGB, RGBA

Mapping a Texture

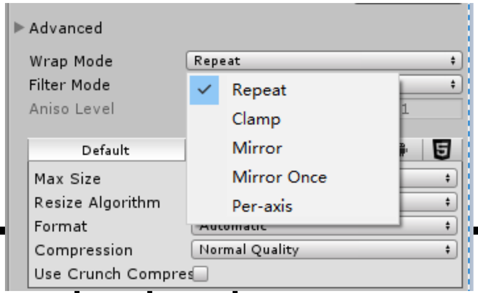
- Based on parametric texture coordinates
- Texture coordinates must be specified for each vertex



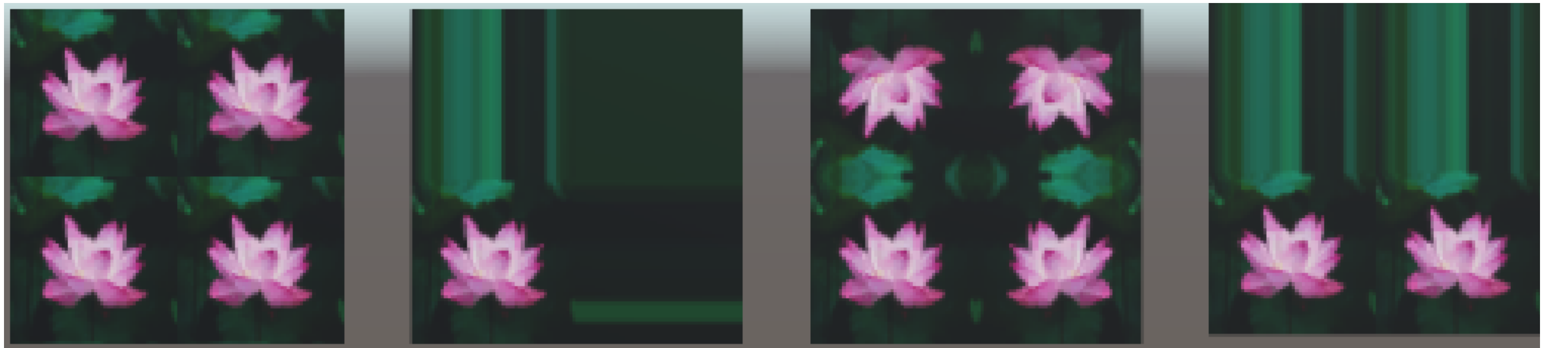
Texture Parameters

- There are a variety of parameters that determine how texture is applied
  - Wrapping – Wrapping parameters determine what happens if u and v are outside the (0,1) range, e.g. Repeat, Clamp, Mirror
  - Filter modes – Filter modes allow us to use area averaging instead of point samples
  - Mipmapping – Mipmapping allows us to use textures at multiple resolutions

Texture Wrap Mode

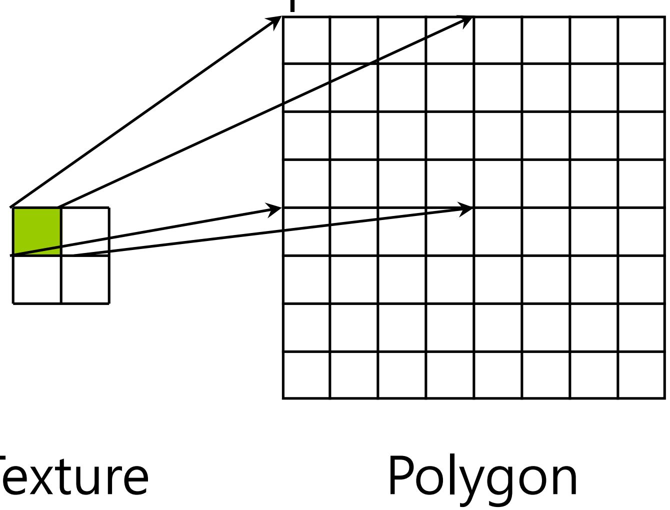


- Wrap mode determines how texture is sampled when **texture coordinates are outside** of the typical 0-1 range.
- In Unity, the texture wrap mode can be
  - **Repeat:** Tiles the texture, creating a repeating pattern
  - **Clamp:** Clamps the texture to the last pixel at the edge
  - **Mirror:** Tiles the texture, creating a repeating pattern by mirroring it at every integer boundary.
  - **MirrorOne:** Mirrors the texture once, then clamps to edge pixels.
  - **Per-axis:** Lets you set different wrap modes for the U axis and the V axis. The available options are also Repeat, Clamp, Mirror and Mirror Once.

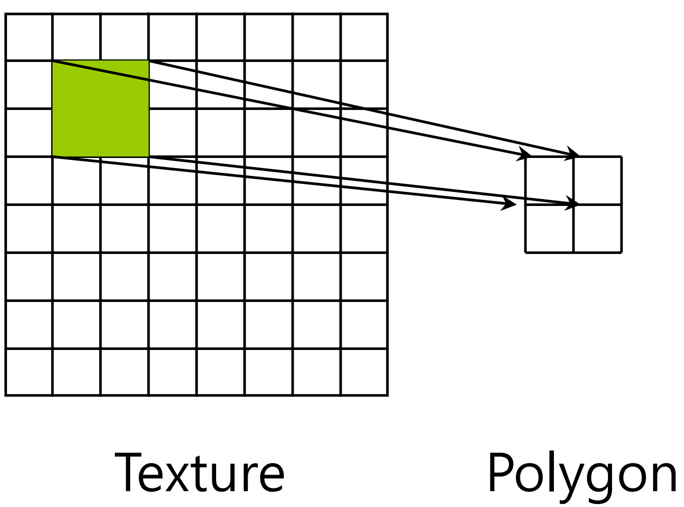


Magnification and Minification

- More than one texel can cover a pixel (**Minification**)
- More than one pixel can cover a texel (**Magnification**)
- Can use point sampling or linear filtering
  - linear filtering – use the weighted average of texel groups including neighbors of texels determined by point sampling
  - Point – use the nearest texel value to the value calculated by line interpolation



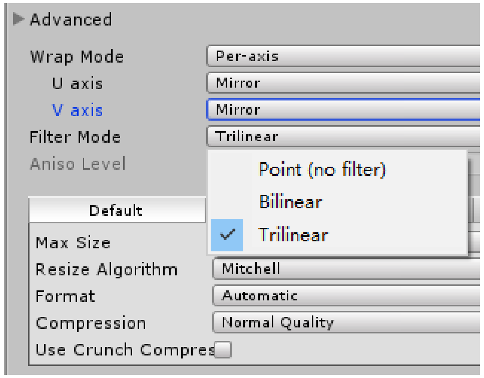
Magnification



Minification

Texture Filtering

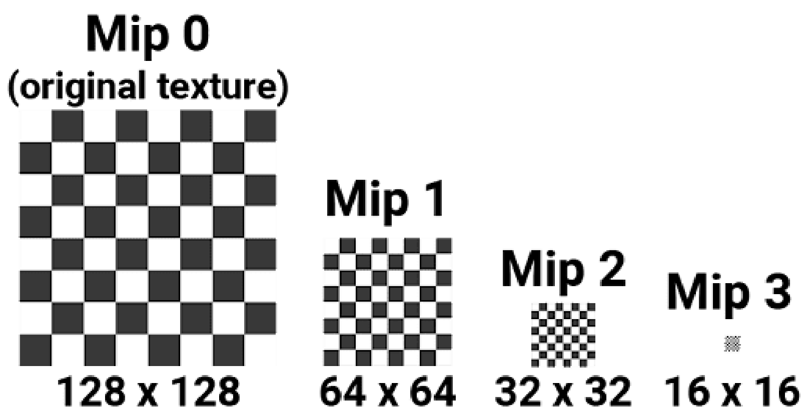
- In Unity, filter mode controls how the sampling of the texture uses nearby pixels.
  - **Point**: uses the **nearest** pixel. This makes the texture appear pixelated.
  - **Bilinear:** uses a **weighted average** of the four nearest **texels**. This makes the texture appear blurry when you magnify it.
  - **Trilinear:** uses a **weighted average** of the two nearest **mips**, which are bilinearly filtered. This creates a soft transition between mips, at the cost of a slightly more blurry appearance



http://www.tomshardware.com/reviews/ati,819-4.html

Mipmaps

- Mipmaps
  - A mip level is a version of a texture with a specific resolution. Mips exist in sets called **mipmaps**. Mipmaps contain progressively smaller and lower resolution versions of a single texture.
  - Mipmaps are commonly used for rendering objects in 3D scenes, where textured objects can vary in distance from the camera. A higher mip level is used for objects closer to the camera, and lower mip levels are used for more distant objects.



https://docs.unity3d.com/Manual/texture-mipmaps-introduction.html
http://www.tomshardware.com/reviews/ati,819-2.html

Anisotropic Filtering

- Anisotropic filtering
  - **Anisotropic filtering** increases texture quality when viewed from a **grazing angle**. This rendering is **resource-intensive** on the graphics card. Increasing the level of anisotropy is usually a good idea for ground and floor Textures.
  - In Unity, use Quality settings to force anisotropic filtering **for all Textures or disable it completely**. Although, if a texture has its **Aniso level set to 0** in Texture Import Settings, forced anisotropic filtering does not appear on this texture.

https://en.wikipedia.org/wiki/Anisotropic_filtering

Texture Coordinate (UV) Transformation

- In Unity, texture coordinate **offset, scaling**, and **rotation** are techniques used to manipulate the way a texture is applied to a 3D object's surface.
  - **Texture Offset** allows you to *shift the position of the texture* on the object's surface. In the shader, the material **SetTextureOffset** function is used to modify the texture offset.
  - **Texture Scaling** adjusts *the size of the texture* on the object's surface. In the shader, the material **SetTextureScale** function is used to change the texture scale.
  - **Texture Rotation** allows you to *rotate the texture* on the object's surface. Rotation is more complex and often involves using a rotation matrix to transform the texture coordinates. This matrix is typically set as a shader property.

Texture Movies

- To create flipbook animation using texture image sequence
  - In Unity, Start() function reads the entire texture images.
  - In Unity, Update() function updates **currentTextureIndex** & sets material's texture using **currentTextureIndex** to the same vertex coordinates and texture coordinates – giving animation effects.

image1

image2

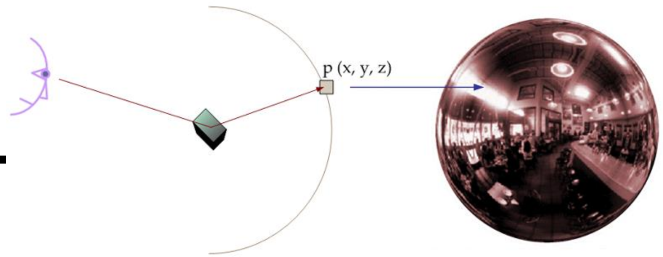
image3

image4

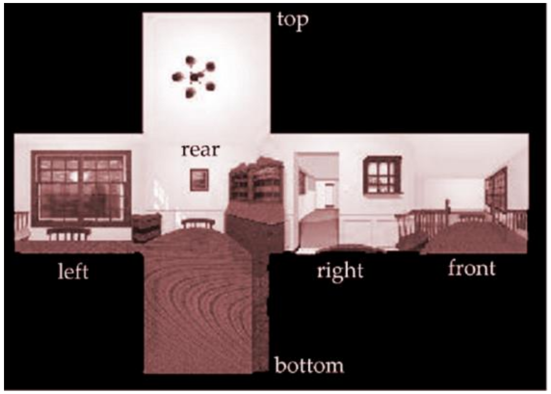
image5

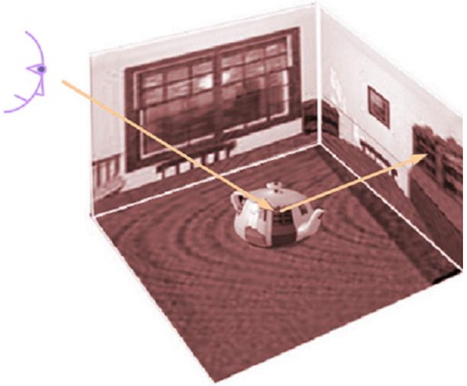
image6

Environment Mapping



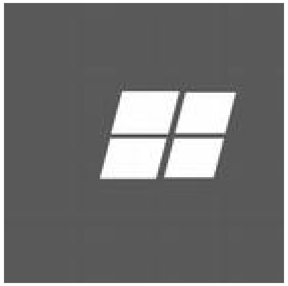
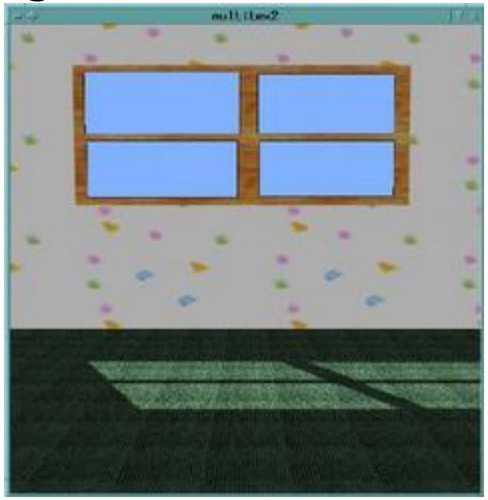
- Environment Maps
  - Start with image of environment through a wide angle lens
  - Use this texture to generate a spherical or box map
  - Use automatic texture coordinate generation
  - Spherical environment mapping – Using automatic texture coordinate generation after creating a spherical map from an environment image taken with a 180 degree wide angle lens



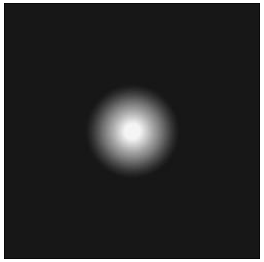
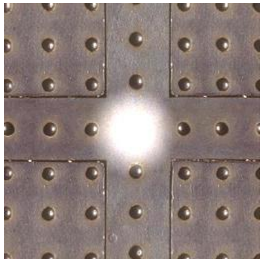


Multitexturing

- Multitexturing
  - Apply a sequence of textures through cascaded texture units


+

=


- Light Mapping
  - Instead of calculating the light of the object surface, the texture and bright image are mixed and the resulting image is directly applied to the object surface


+

=


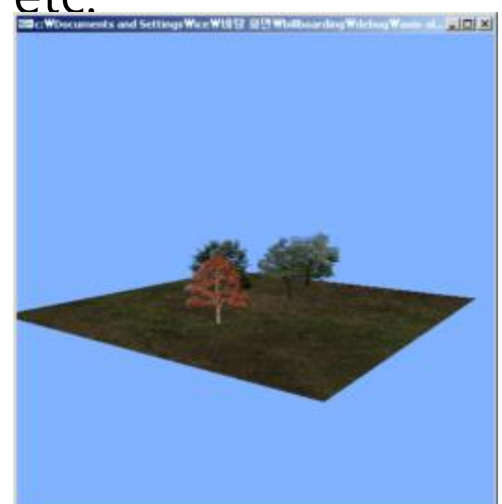
Multitexturing

- Single-Pass vs. Multi-Pass Multitexturing
  - Single-pass multitexturing means applying multiple textures within one rendering pass.
  - Multi-pass multitexturing is the rendering of the scene or the polygon itself multiple times by blending.

Billboarding

- Billboard technique
  - The front of billboard rectangle is made to always look toward the camera, and as a result, the billboard always shows the same side no matter which direction the camera is viewed.
  - For example, tree billboard images are used to create a forest, instead of using tree mesh models.
  - The billboard technique combined with the alpha texture is used to express various natural phenomena that do not have a solid surface: smoke, fire, fog, explosion, etc.

Billboarding

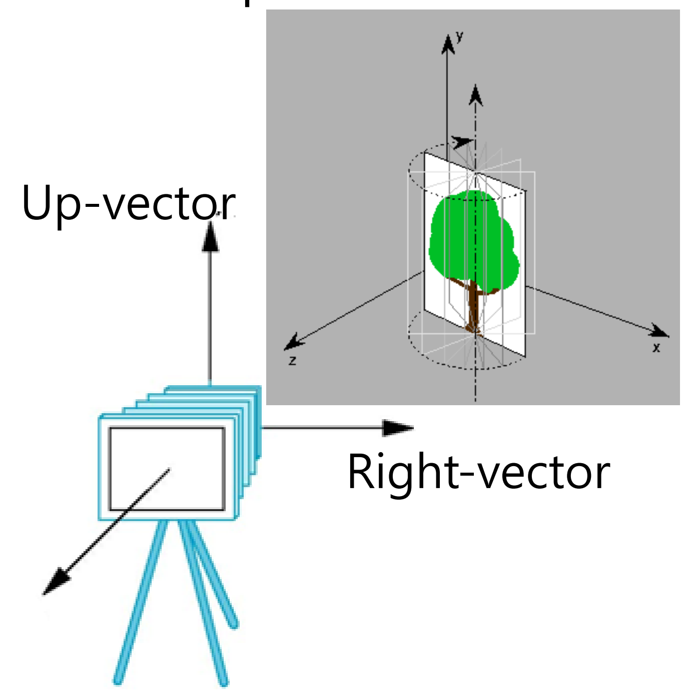


Billboarding

- Billboarding principle
  - The key to implementation is to adjust the vertices that make up the billboard square using the modelview matrix so that the user always look at the viewpoint.
  - The modelview matrix contains information about the up vector and the right vector of the viewer's viewpoint.

Right-vector Up-vector Look-vector

$$\begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix}$$



Billboarding

- Axial Symmetry
  - Billboard rectangle should rotate around the vertical axis.
  - Calculate the camera's yaw angle from the Modelview matrix, M.

```
theta = atan2f(M[8], M[10]);
                Look.x    Look.z
```

  - The rotation matrix, R, of the billboard rectangle is calculated as an arbitrary axis (typically, up vector=(0, 1, 0)) and angle (inverse of the camera yaw angle).

$$R = \mathrm{I}\cos\theta + \mathbf{Symmetric}\ (1-\cos\theta) + \mathbf{Skew}\ \sin\theta$$

$$= \begin{bmatrix} a_x^2 + \cos\theta(1-a_x^2) & a_x a_y(1-\cos\theta) - a_z\sin\theta & a_x a_z(1-\cos\theta) + a_y\sin\theta \\ a_x a_y(1-\cos\theta) + a_z\sin\theta & a_y^2 + \cos\theta(1-a_y^2) & a_y a_z(1-\cos\theta) - a_x\sin\theta \\ a_x a_z(1-\cos\theta) - a_y\sin\theta & a_y a_z(1-\cos\theta) + a_x\sin\theta & a_z^2 + \cos\theta(1-a_z^2) \end{bmatrix}$$