Textures

Logistics

- Programming Assignment 1 – Line Drawing
- Programming Assignment 2 – Polygon Fill
- Programming Assignment 3 – Clipping
- Programming Assignment 4 – Hello OpenGL
  - All done including resubmits
- Programming Assignment 5 - Tesselation
  - Resubmits due 2/4.
- Programming Assignment 6 – Transformations / Viewing
  - More about this.
- Programming Assignment 7 – Shading
  - Due Thursday.

Logistics

- Grad report
  - List of papers due Friday, Jan 11th
    - All done
  - Actual Report due February 15th

Plan for this week

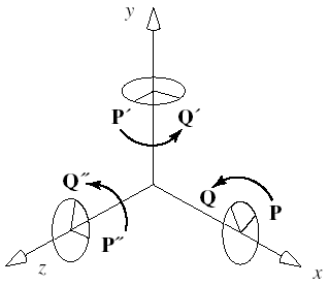
- Texture Mapping
- Today
  - Concepts
  - About the midterm
- Thursday
  - Doing it in OpenGL
  - + Final exam.
- But first..

About the transformation assignment

- Rotations
  - Must consider
    - Right handed vs Left Handed coordinate systems
    - Row-major vs. Column-major matrix representation
    - Pre-multiply or Post-multiply

RHS vs LHS

- Right handed coordinate system
  - Positive rotation is counterclockwise
  - +Z extends toward viewer
  - Used by OpenGL



Right handed CS

- Left handed coordinate system
  - Positive rotation is clockwise
  - +Z extends toward screen
  - Used by DirectX

Row-order vs. column-order

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix}$$

Row major order
DirectX

$$\begin{bmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{bmatrix}$$

Column major order
OpenGL

Row-order vs. column-order

- Translation

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Row major order

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

Column major order

Row-order vs. column-order

- Rotation – ROW MAJOR ORDER

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotate around x | Rotate around y | Rotate around z

Row-order vs. column-order

- Rotation – COLUMN MAJOR ORDER

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotate around x | Rotate around y | Rotate around z

Pre-multiply / post-multiply

To apply transformation A followed by transformation B

$$v_{new} = BAv_{old}$$

Post-multiply
(vertex coords)
OpenGL

$$v_{new} = v_{old}AB$$

Pre-multiply
(vertex coords)
DirectX

post-multiplying with column-major matrices produces the same result as pre-multiplying with row-major matrices.

Matrix organization and multiplication

$$\begin{bmatrix} x & y & z & w \end{bmatrix} = \begin{bmatrix} x_o & y_o & z_o & w_o \end{bmatrix} \begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{bmatrix}$$

1 x 4 = 1 x 4 × 4 x 4

Row major
point as row vector
Pre-multiply

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ w_o \end{bmatrix}$$

4x1 = 4 x 4 × 4x1

Col major
point as col vector
Post-multiply

http://scratchapixel.com/lessons/3d-basic-lessons/lesson-4-geometry/conventions-again-row-major-vs-column-major-vector/

CORRECT assignment params

- **Model Transforms** -- The following transformations should be applied to the object, **in this order:**
  - Scale in the y direction by a factor of 5
  - Rotate around the z axis by 90 degrees
  - Rotate around the y axis by **30** degrees
  - Translate by -1.0 in the z direction and **1.0** in the x
- **View Transform** -- The parameters for the camera should be set as follows:
  - Camera should be located at (0.0, 3.0, 3.0)
  - Camera should be looking at (1.0, 0.0, 0.0)
  - The up vector for the camera should be (0.0, 1.0, 0.0)
- **Projection** -- Both frustrum and orthographic projection will be implemented. For both, the parameter values for the projection should be:
  - Left = -1.5
  - Right = 1.0
  - Top = 1.5
  - Bottom = -1.5
  - Near = 1.0
  - Far = 8.5

Transformation Assignment

- Dropbox reopened until end of quarter!
- Questions?

Texture Mapping Slides

- Rosalie Wolfe
  - SIGGRAPH 97
  - http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_1.htm

Texture Mapping

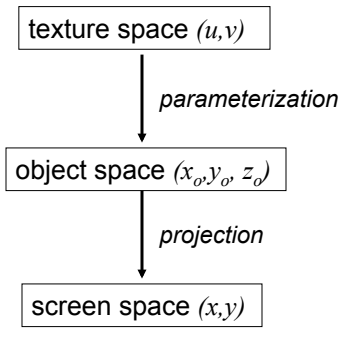
- Developed in 1974 by Ed Catmull
  - Currently, president of Pixar (head of Disney Animation)
- Goal: to make Phong shading less plastic-looking
- Uses of texturing
  - Simulating materials
  - Reducing geometric complexity
  - Image warping

Texture Mapping

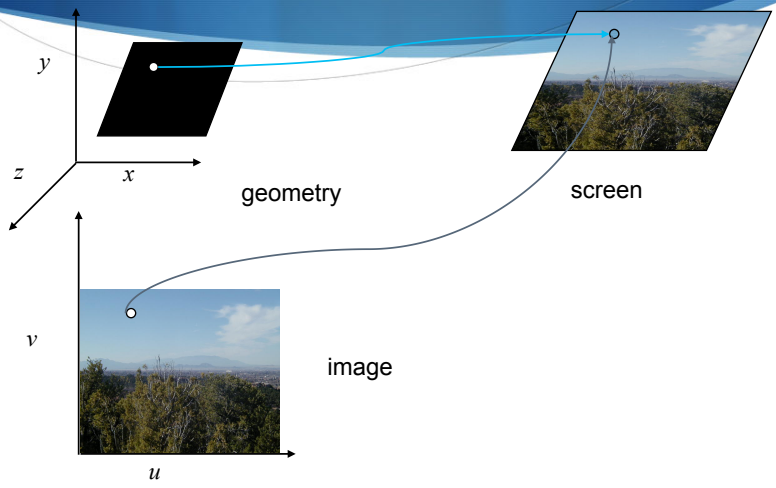
- Two basic types of textures:
  - Image files
  - Mathematically-generated (e.g., checkerboard pattern)
- Textures made up of *texture elements* (texels)
- Can be 1D, 2D, or 3D
  - 1D textures are images with dimension 1xN or Nx1
  - 2D textures have width and height
  - 3D textures describe volumes
    - Used, e.g., in CT/MRI scans
    - Huge – a 256x256x256 grayscale-alpha texture uses 32MB of memory
- For simplicity, we'll mostly look at 2D textures
  - Same principles apply to 1D and 3D

Texture Mapping

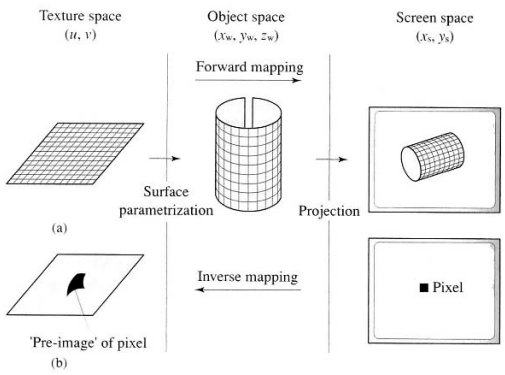
- Define surface characteristics of an object using an image
- Basic concept: associate value at some coordinates in texture with pixel at some coordinates in geometry
- Coordinate spaces used in texture mapping:
  - Texture space *(u, v)*
  - Object space (x_o, y_o, z_o)
  - Screen space *(x, y)*



Texture Mapping

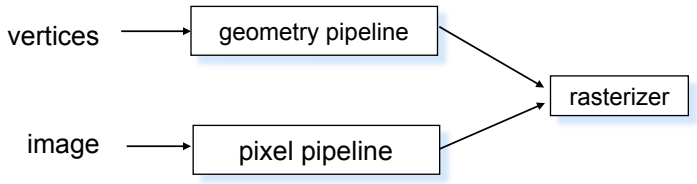


Texture Mapping



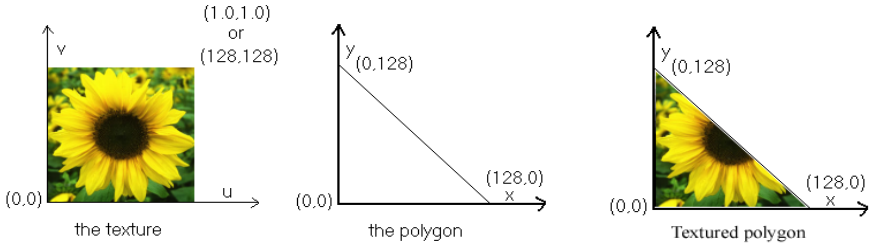
Texture Mapping and the OpenGL Pipeline

- Images and geometry flow through separate pipelines that join at the rasterizer
- Advantage: visual detail is in the image, not the geometry
  - "Complex" textures do not affect geometric complexity



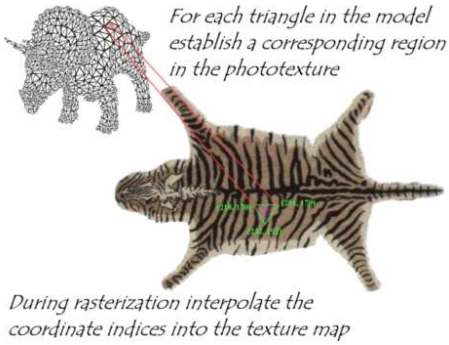
Direct Mapping

- Simplest form of mapping
- Texture coordinates equal polygon coordinates. ($u = x$ and $v = y$)
- How to fill? Polygon filling
  - Same scanline algorithm as used before
  - Color data not constant – instead, pulled from texture array



Direct Mapping

- Photographic textures

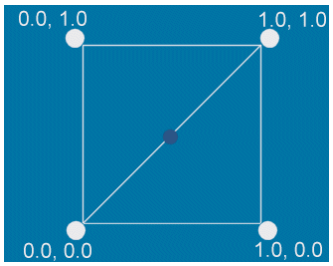


Direct Mapping

- Problems:
  - Causes texture to be clipped, or entire polygon not filled
- Need an algorithm that stretches and shrinks texture over a polygon
- Affine mapping comes to the rescue

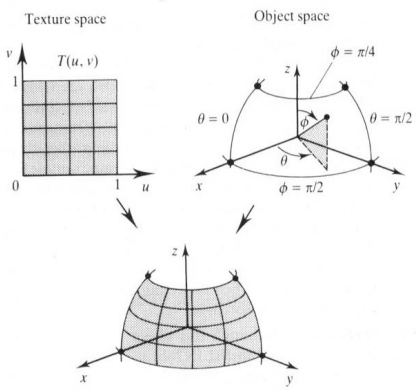
Affine Mapping

- Stretching and shrinking of the texture is based on the premise of sampling
- Texture coordinates become necessary
  - Arbitrary mapping of texture coordinates to polygon coordinates

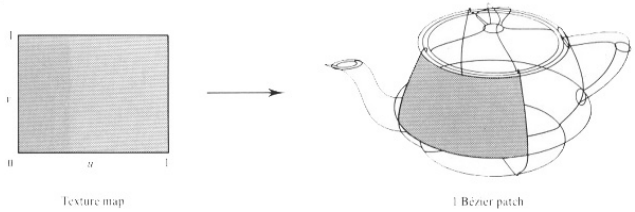


Affine Mapping

- Texture is like a rubber sheet stretched to fit model

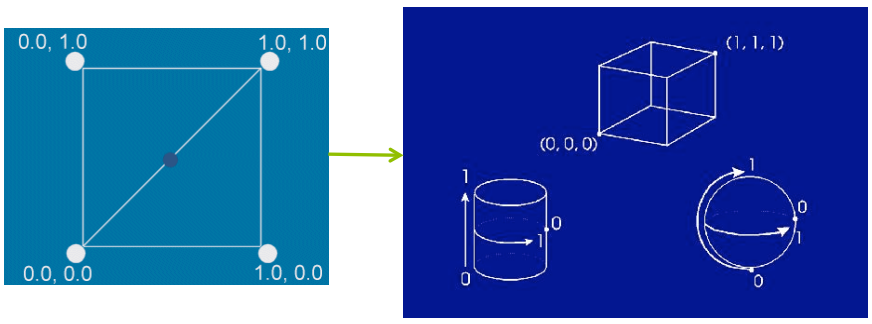


Affine Mapping

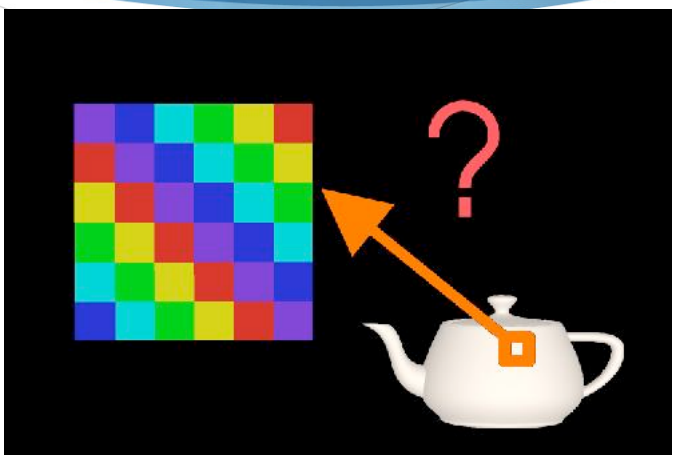


Watt

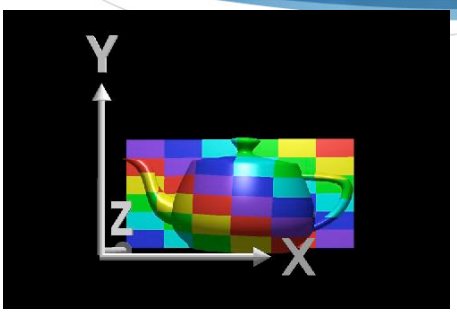
Types of Mapping

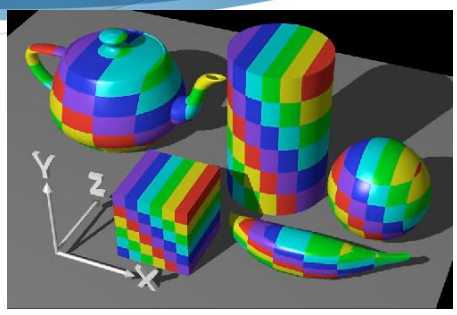


Texturing



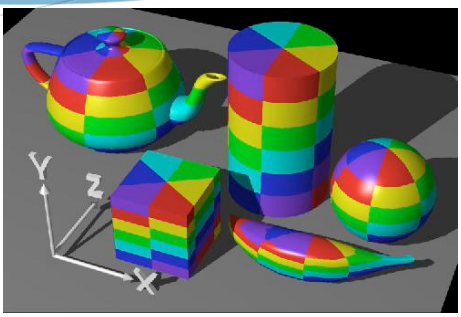
Planar Map





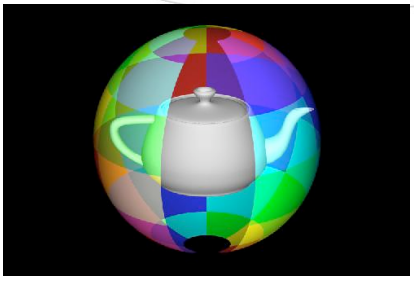
http://www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/textureMapping/tiled_planar_texture_mapping_guide.html

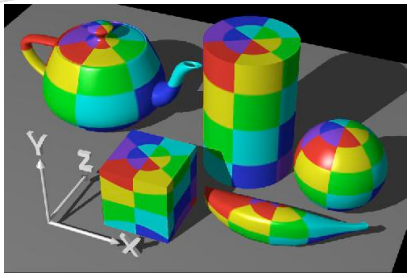
Cylindrical Mapping



http://www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/textureMapping/cylindrical_texture_mapping_guide.html

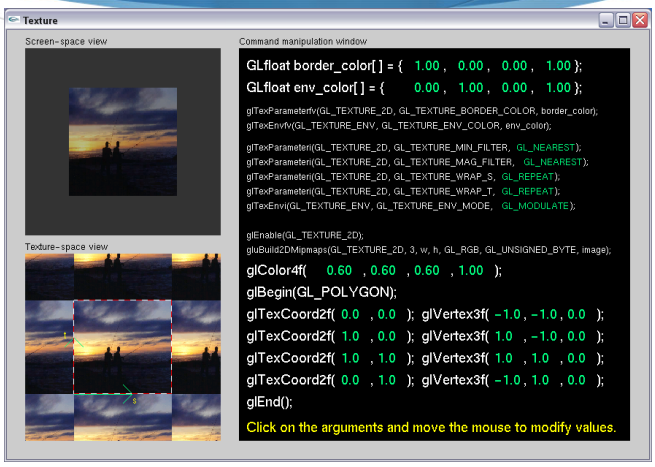
Spherical Mapping



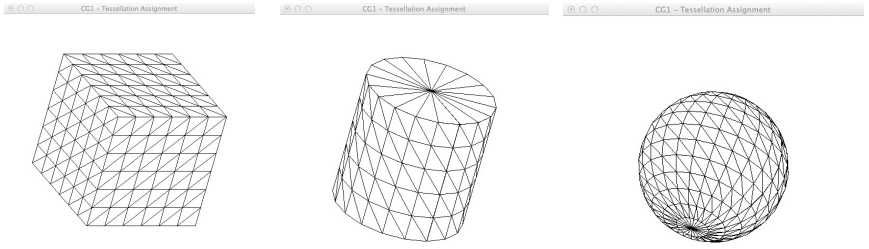


http://www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/textureMapping/spherical_texture_mapping_guide.html

Texture Coordinates



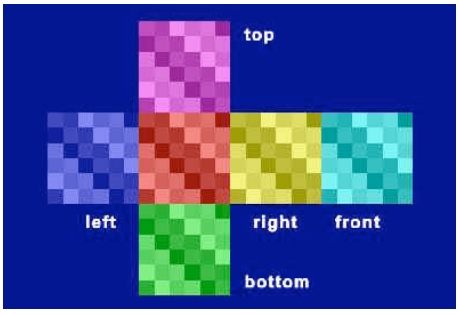
Texturing and Tessellation

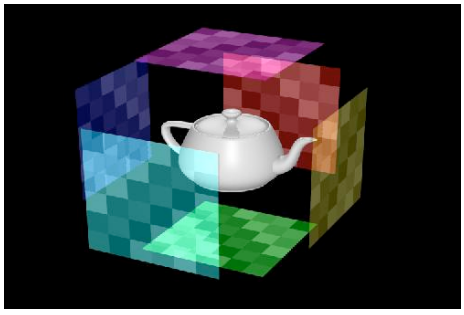


Texture as data

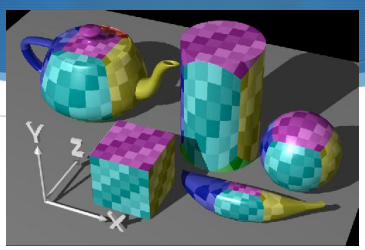
- In the "new" OpenGL
  - Textures can be accessed in the pixel shader
  - Simply data attached to an object at a given point
  - Exploits texture memory on graphics card
  - Can be used for any purpose.

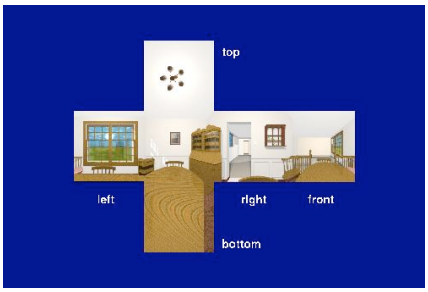
Cube Mapping





Cube / Reflection Mapping



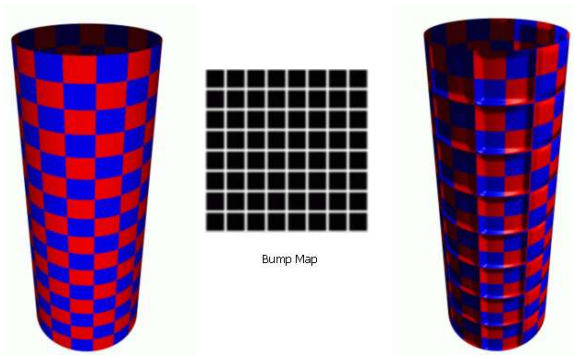


Texture Mapping – Bump Mapping

- Developed by Jim Blinn
- Adds roughness to surfaces
- Quick way to add detail to an object
- Polygon remains physically flat, but appears bumpy

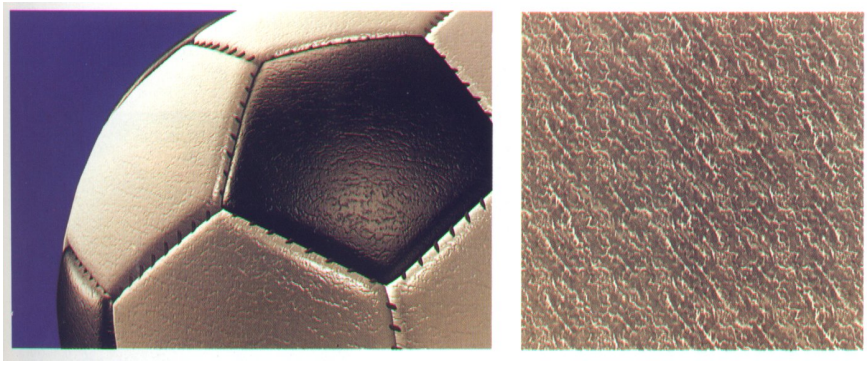
Bump Mapping

- Bump map provides additional surface detail



Bump Mapping

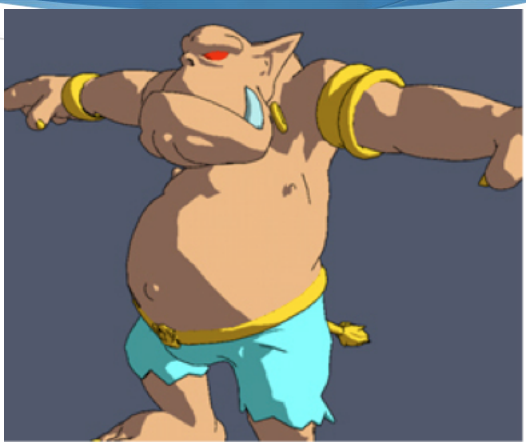
- More realistic surface representation



Toon Shading

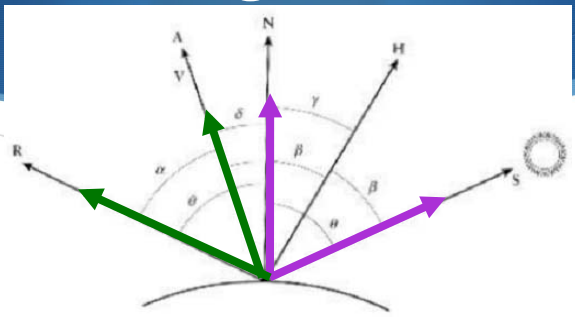
- Mimic the look of cartoons
  - Intentionally 2-dimensional
  - Use of a set solid colors (rather than a gradiant)
  - Hard edges

Toon Shading



[Lake, et. al. 2000]

Phong Model



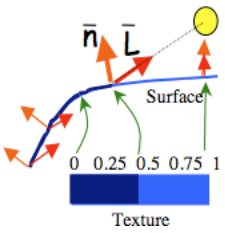
$$L(V) = k_a L_a + k_d \sum_i L_i (\mathrm{S}_i \bullet \mathrm{N}) + k_s \sum_i L_i (\mathrm{R}_i \bullet \mathrm{V})^{k_e}$$

ambient diffuse specular

Note: L_n are radiance terms, include both light and material info

Toon Shading

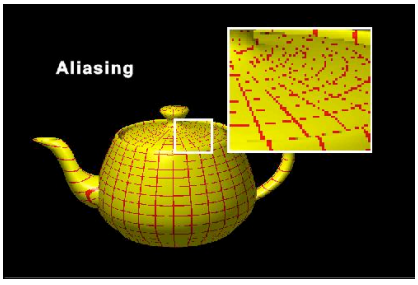
- Much like diffuse shading but…
  - Rather than using N • L to determine shade color
  - Use N • L as index to a 1D texture map.

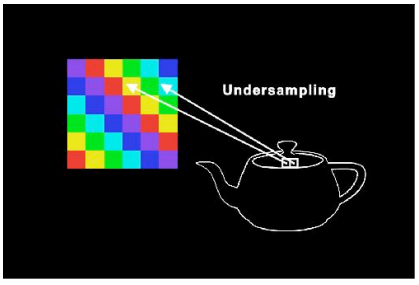


Toon Shading

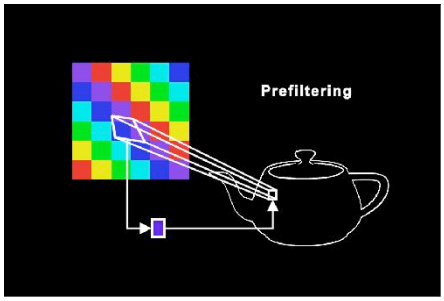
- To avoid aliasing
  - Use large map with gradient between color transitions.

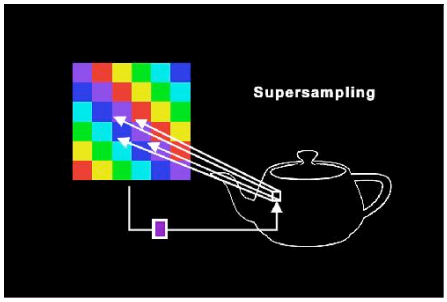
Texture Mapping and Aliasing



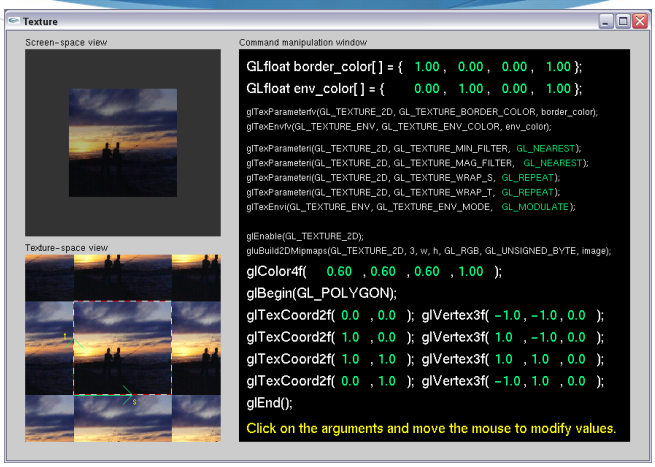


Texture Mapping / Antialiasing



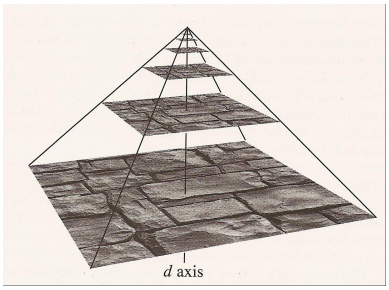


Texture Filtering



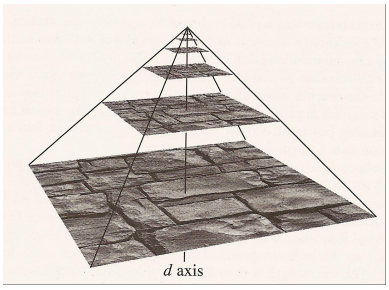
Texture Mapping – Level of Detail

- Can use *multiple image patterns* (mipmaps)
  - Pre-calculate your texture map at may resolutions (or layers)
  - Store all "texture layers" in a single image
  - Use "appropriate" layer when rendering, interpolating between levels as required

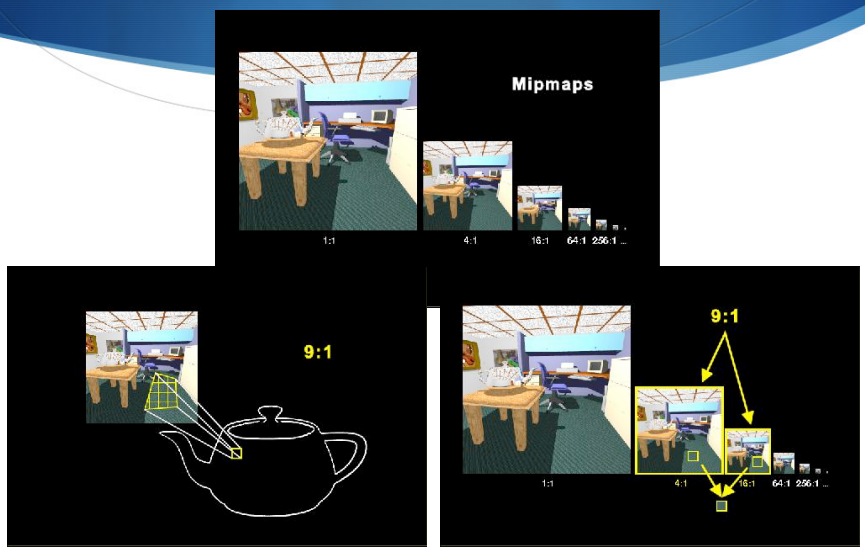


Texture Mapping – Level of Detail

- Can use *multiple image patterns* (mipmaps)
  - Pre-calculate your texture map at may resolutions (or layers)
  - Store all "texture layers" in a single image
  - Use "appropriate" layer when rendering, interpolating between levels as required



Mipmaps



Summary

- Texture Mapping
  - Use of image to add complexity
  - 2D -> 3D -> 2D Mapping
  - Bump Mapping
  - Environment Mapping
  - Textures and Shaders

- Questions?