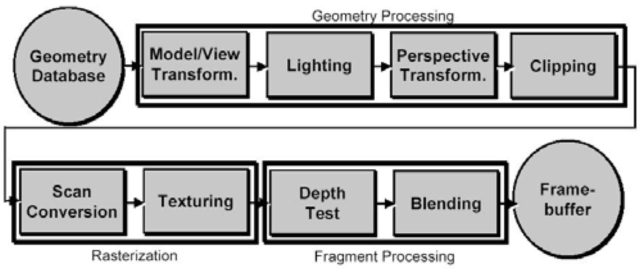
Texture Mapping

CPSC 414
10/24/03

Abhijeet Ghosh

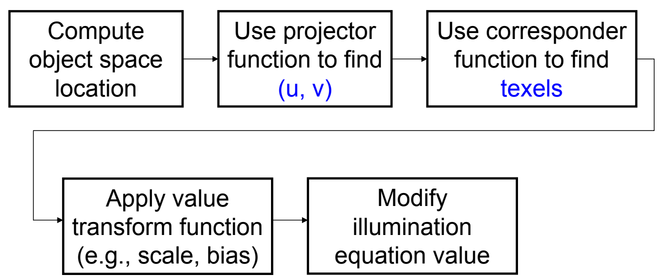
The Rendering Pipeline



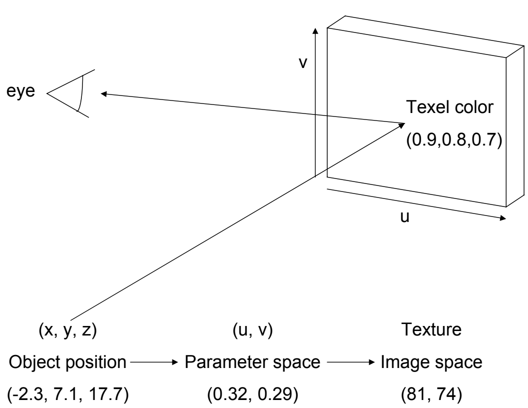
Texture Mapping

- Associate 2D information with 3D surface
  - Point on surface corresponds to a point in the texture
- Introduced to increase realism
  - Lighting/shading models not enough
- Hide geometric simplicity
  - map a brick wall texture on a flat polygon
  - create bumpy effect on surface

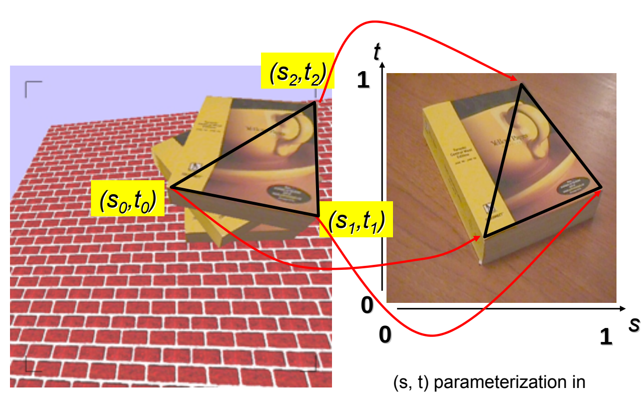
Texture Pipeline



Texture Pipeline



Texture Mapping



(s, t) parameterization in OpenGL

Texture Mapping

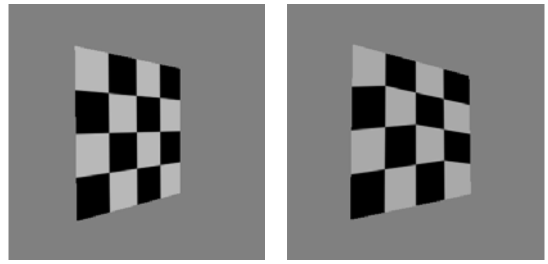
- Texture Coordinates
  - generation at vertices
    - specified by programmer or artist

      ```
      glTexCoord2f(s,t)
      glVertexf(x,y,z)
      ```
    - generate as a function of vertex coords

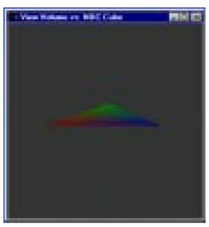
      ```
      glTexGeni(), glTexGenfv()
      ```
      *s = a*x + b*y + c*z + d*h*
  - interpolated across triangle (like R,G,B,Z)
    - …well not quite!

Texture Mapping

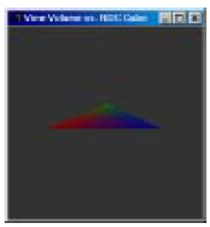
- Texture Coordinate Interpolation
  - perspective foreshortening problem
  - also problematic for color interpolation, etc.



Attribute Interpolation



Bilinear Interpolation
Incorrect



Perspective correct
Correct

Texture Coordinate Interpolation

- Perspective Correct Interpolation
  - α, β, γ :
    - Barycentric coordinates of a point **P** in a triangle
  - *s0*, *s1*, *s2* :   texture coordinates
  - *w0*, *w1*,*w2* : homogeneous coordinates

$$s = \frac{\alpha \cdot s_0 / w_0 + \beta \cdot s_1 / w_1 + \gamma \cdot s_2 / w_2}{\alpha / w_0 + \beta / w_1 + \gamma / w_2}$$

Texture Mapping

- Textures of other dimensions
  - 3D: solid textures
    - e.g.: wood grain, medical data, ...
    - **glTexCoord3f(s,t,r)**
  - 4D: 3D + time, projecting textures
    - **glTexCoord3f(s,t,r,q)**

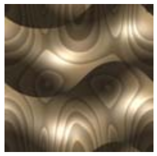
Texture Coordinate Transformation

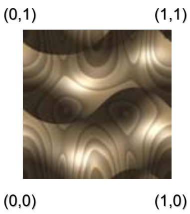
- Motivation:
  - Change scale, orientation of texture on an object
- Approach:
  - *texture matrix stack*
  - 4x4 matrix stack
  - transforms specified (or generated) tex coords

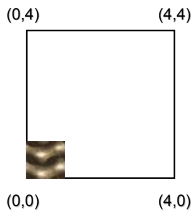
    ```
    glMatrixMode( GL_TEXTURE );
    glLoadIdentity();
      …
    ```

Texture Coordinate Transformation

- Example:







`glScalef(4.0,4.0,?);`

Texture Lookup

- Issue:
  – What happens to fragments with *s* or *t* outside the interval [*0…1*]?

Multiple choices:

– Take only fractional part of texture coordinates
  - Cyclic repetition of texture to tile whole surface

    glTexParameteri(…, GL_TEXTURE_WRAP_S, GL_REPEAT)

– Clamp every component to range [0…1]
  - Re-use color values from border of texture image

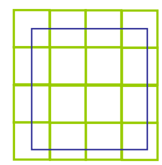
    glTexParameteri(…, GL_TEXTURE_WRAP_S, GL_CLAMP)

Texture Functions

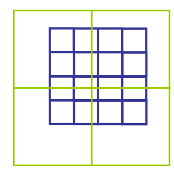
- Once got value from the texture map, can:
  – Directly use as surface color `GL_REPLACE`
  – Modulate surface color `GL_MODULATE`
  – Blend surface and texture colors `GL_DECAL`
  – Blend surface color with another `GL_BLEND`
- Specific action depends on internal texture format
  – Only existing channels used
- Specify with `glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, mode)`

Reconstruction

– How to deal with:
  - pixels that are much larger than texels ?
    – apply filtering, "averaging"

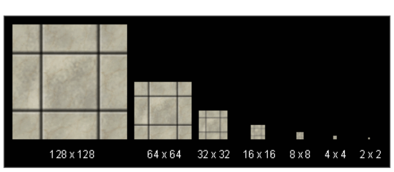


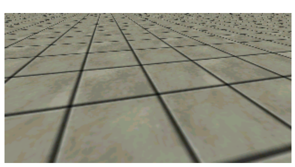
  - pixels that are much smaller than texels ?
    – interpolate



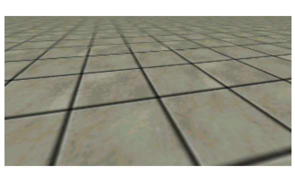
Mip-mapping

Use an "image pyramid" to pre-compute averaged versions of the texture





Without MIP-mapping

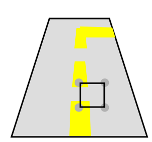


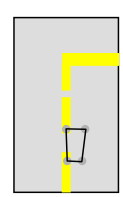
With MIP-mapping

Mip-mapping

Problem:

- A MIP-map level selects the same minification factor for both the *s* and the *t* direction (isotropic filtering)
- In reality, perspective foreshortening (amongst other reasons) can cause different scaling factors for the two directions





Mip-mapping

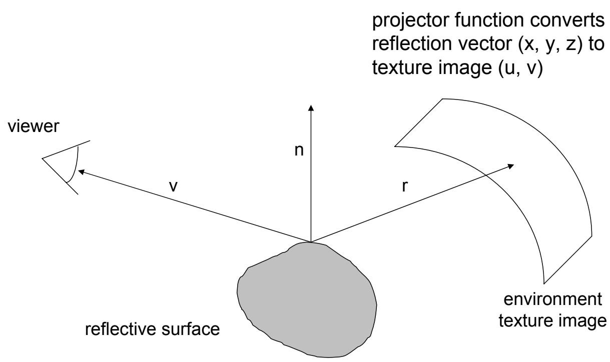
- Which resolution to choose:
  - MIP-mapping: take resolution corresponding to the smaller of the sampling rates for *s* and *t*
    - Avoids aliasing in one direction at cost of blurring in the other direction
  - Better: anisotropic texture filtering
    - Also uses MIP-map hierarchy
    - Choose larger of sampling rates to select MIP-map level
    - Then use more samples for that level to avoid aliasing
    - Maximum anisotropy (ratio between s and t sampling rate) usually limited (e.g. 4 or 8)

Texture Mapping Functions

Two Step Parameterization:

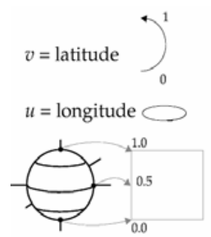
- Step 1: map 2D texture onto an intermediate simple surface
  - Sphere
  - Cube
  - Cylinder
- Step 2: map from this surface to the object
  - Surface normal
- Commonly used for environment mapping

Environment Mapping



Spherical Maps – Blinn & Newell '76

- Transform reflection vector *r* into spherical coordinates *(θ, Φ)*
  - *θ* varies from [0, π] (latitude)
  - *Φ* varies from [0, 2π] (longitude)



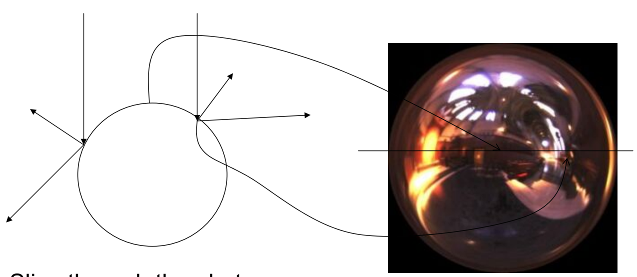
$r = (r_x, r_y, r_z) = 2(n.v)n - v$

$\Theta = \arccos(-r_z)$

$\Phi = \{ \arccos(-r_x/\sin\Theta)$ if $r_y \geq 0$

$\{ 2\pi - \arccos(-r_x/\sin\Theta)$ otherwise

Spherical Maps – Blinn & Newell '76

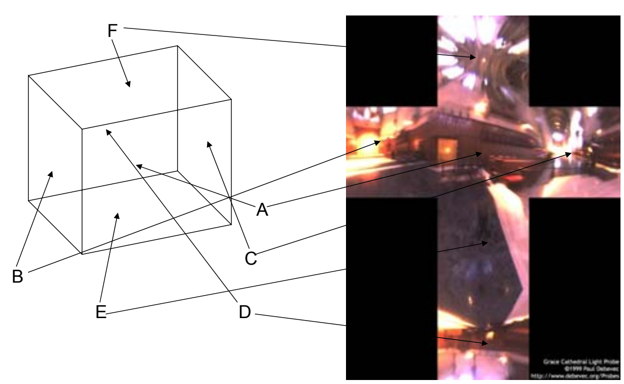


Slice through the photo

Each pixel corresponds to particular direction in the environment

- Singularity at the poles!
- OpenGL support `GL_SPHERE_MAP`

Cube Mapping – Greene '86



Cube Mapping – Greene '86

- Direction of reflection vector *r* selects the face of the cube to be indexed
  - Co-ordinate with largest magnitude
    - e.g., the vector (-0.2, 0.5, -0.84) selects the –Z face!
  - Remaining two coordinates (normalized by the 3rd coordinate) selects the pixel from the face.
    - e.g., (-0.2, 0.5) gets mapped to (0.38, 0.80).
- Difficulty in interpolating across faces!
- OpenGL support `GL_CUBE_MAP`