Texture Filtering

MipMaps

“Optimal” case

Minification

Magnification

Magnification and Minification
More than one texel can cover a pixel (minification) or more than one pixel can cover a texel (magnification)

Can use point sampling (nearest texel) or linear filtering (2 x 2 filter) to obtain texture values

Pixel Footprint
Pyramid Textures (Mipmapping)

Linear vs. Nearest

Trilinear
Mipmapped Textures

- **Mipmapping** allows for prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
- Declare mipmap level during texture definition
  \[ \text{glTexImage2D( GL_TEXTURE\_D, level, ... )} \]
- GLU mipmap builder routines will build all the textures from a given image
  \[ \text{gluBuild\_*D\_Mipmaps( ... )} \]

Example

![Mipmapped Textures Example](image)

Anisotropic Filtering

![Anisotropic Filtering](image)

Anisotropic Filtering

![Anisotropic Filtering](image)

Anisotropic Filtering

![Anisotropic Filtering](image)
Light Mapping

• In order to keep the texture and light maps separate, we need to be able to perform multitexturing – application of multiple textures in a single rendering pass

• How do you create light maps?
  • Trying to create a light map that will be used on a non-planar object things get complex fast:
    – Need to find a divide object into triangles with similar orientations
    – These similarly oriented triangles can all be mapped with a single light map

• Things for standard games are usually much easier since the objects being light mapped are usually planar:
  – Walls
  – Ceilings
  – Boxes
  – Tables
• Thus, the entire planar object can be mapped with a single texture map

• Can dynamic lighting be simulated by using a light map?
  • If the light is moving (perhaps attached to the viewer or a projectile) then the lighting will change on the surface as the light moves
    – Moving flashlight (use texture matrix)
    – The light map values can be partially updated dynamically as the program runs
    – Several light maps at different levels of intensity could be pre-computed and selected depending on the light’s distance from the surface

Lightmaps

• Creating local contribution

Unlit Scene  Lightmap Intensity  Local Light Contribution  Lightmap
Lightmaps

- Adding local light to scene
  
  ![OpenGL Lighting](image1)
  ![Combined Image](image2)

  **Demo**

Lightmaps

- Cached Lighting Results
  - Reuse lighting calculations
    - Multiple local lights (same type)
    - Static portion of scene’s light field
    - Sample region with texture instead of tessellating
  - Low resolution sampling
    - Local lighting: rapid change over small area
    - Global lighting: slow change over large area

Lightmaps

- Segmenting Scene Lighting
  - Static vs. dynamic light fields
  - Global vs. local lighting
  - Similar light shape

Lightmaps

- Segmenting the lighting
  
  ![Dominant Lighting](image3)
  ![Local lighting](image4)

Lightmaps

- Moving Local Lights
  - Recreate the texture; simple but slow
  - Manipulate the lightmap
    - Translate to move relative to the surface
    - Scale to change spot size
    - Change base polygon color to adjust intensity
  - Projective textures ideal for spotlights
  - 3D textures easy to use (if available)

Lightmaps

- Special Case
  
  **Spotlights as Lightmap**

  - Mapping Single Spotlight Texture Pattern
    
    ![Original](image5)
    ![Sequential Texture](image6)
    ![Gradient Texture](image7)
    ![Lightmap Texture](image8)

    Use texture matrix to perform spotlight texture coordinates transformations.
Lightmaps

• Creating a lightmap
  – Light white, tesselated surface with local light
  – Render, capture image as texture
  – Texture contains ambient and diffuse lighting
  – glLight() parameters should match light
  – Texture can also be computed analytically

Lightmaps

• Lightmap building tips
  – Boundary should have constant value
  – Intensity changes from light should be minimal near edge of lightmap

Lightmaps

• Lighting with a Lightmap
  – Local light is affected by surface color and texture
  – Two step process adds local light contribution:
    • Modulate textured, unlit surfaces with lightmap
    • Add locally lit image to scene
  – Can mix OpenGL, lightmap lighting in same scene

Lightmaps

• Creating local contribution

Lightmaps

• Adding local light to scene
**Lightmaps in Quake2**

- Lightmaps only
- Decal only
- Combined scene

**Packing Many Lightmaps into a Single Texture**

- Quake 2 light map texture image example
- Lightmaps typically heavily magnified.
- Permits multiple lightmaps packed into a single texture.
- Quake 2 computes lightmaps via off-line radiosity solver.

**Lightmaps**

- Lightmap considerations
  - Lightmaps are good:
    - Under-tessellated surfaces
    - Custom lighting
    - Multiple identical lights
    - Static scene lighting
  - Lightmaps less helpful:
    - Highly tessellated surfaces
    - Directional lights
    - Combine with other surface effects (e.g. bump-mapping)
- Multitexturing
  - Multitexturing allows the use of multiple textures at one time.
  - It is a standard feature of OpenGL 1.3 and later.
  - An ordinary texture combines the base color of a polygon with color from the texture image. In multitexturing, this result of the first texturing can be combined with color from another texture.
  - Each texture can be applied with different texture coordinates.
**Texture Units**

- Multitexturing uses multiple texture units.
- A texture unit is a part of the rendering pipeline that applies one texture to whatever is being drawn.
- Each unit has a texture, a texture environment, and optional texgen mode. That is, its own complete and independent OpenGL texture state.
- Most current hardware has from 2 to 16 texture units.
- To get the number of units available: `glGetIntegerv(GL_MAX_TEXTURE_UNITS)`

**OpenGL Multitexture Quick Tutorial**

- Configuring up a given texture unit:
  - `tex1_uniform_loc = glGetUniformLocation(prog, "tex1");`
  - `glUniform1i(tex1_uniform_loc, 1);`
  - `glActiveTexture(GL_TEXTURE1);`
  - `glBindTexture(GL_TEXTURE_2D, texObject);`
  - `glTexImage2D(GL_TEXTURE_2D, …);`
  - `glTexParameterfv(GL_TEXTURE_2D, …);`
  - `glTexEnvfv(GL_TEXTURE_ENV, …);`
  - `glTexGenfv(GL_S, …);`
  - `glMatrixMode(GL_TEXTURE);`
  - `glLoadIdentity();`

- Setting texture coordinates for a vertex:
  - `glMultiTexCoord4f(GL_TEXTURE0, s0, t0, r0, q0);`
  - `glMultiTexCoord2f(GL_TEXTURE1, s1, t1);`
  - `glMultiTexCoord3f(GL_TEXTURE2, s2, t2, r2);`
  - `glVertex3f(x, y, z);`

**OpenGL Multitexture Texture Environments (old way)**

- Chain of Texture Environment Stages:
  - `glMultiTexCoord2f(GL_TEXTURE0_ARB, …)
  - glMultiTexCoord2f(GL_TEXTURE1_ARB, …)
  - glMultiTexCoord2f(GL_TEXTURE2_ARB, …)

**OpenGL Multitexture Texture Environments (new way)**

- Chain of Texture Environment Stages:
  - `varying vec3 lightDir, normal;`  
  - `void main(){`  
  - `vec3  ct, cf;`  
  - `vec4 texel;`  
  - `float intensity, at, af;`  
  - `intensity = max(dot(lightDir, normalize(normal)),0.0);`  
  - `cf = intensity * gl_FrontMaterial.diffuse.rgb + gl_FrontMaterial.ambient.rgb;`  
  - `af = gl_FrontMaterial.diffuse.a;`  
  - `texel = texture2D(tex0,gl_TexCoord[0].st) + texture2D(tex0,gl_TexCoord[1].st);`  
  - `ct = texel.rgb;`  
  - `at = texel.a;`  
  - `gl_FragColor = vec4(ct*cf, at*af);`  
  - `}`
Multitexture Lightmapping

Alpha Mapping

• An Alpha Map contains a single value with transparency information
  – 0 → fully transparent
  – 1 → fully opaque
• Can be used to make sections of objects transparent
• Can be used in combination with standard texture maps to produce cutouts
  – Trees
  – Torches

Alpha Mapping

• In the previous tree example, all the trees are texture mapped onto flat polygons
• The illusion breaks down if the viewer sees the tree from the side
• Thus, this technique is usually used with another technique called “billboarding”
  – Simply automatically rotating the polygon so it always faces the viewer
• Note that if the alpha map is used to provide transparency for texture map colors, one can often combine the 4 pieces of information (R,G,B,A) into a single texture map

Alpha Mapping

• The only issue as far as the rendering pipeline is concerned is that the pixels of the object made transparent by the alpha map cannot change the value in the z-buffer
  – We saw similar issues when talking about whole objects that were partially transparent → render them last with the z-buffer in read-only mode
  – However, alpha mapping requires changing z-buffer modes per pixel based on texel information
  – This implies that we need some simple hardware support to make this happen properly

Bill Boarding

• How?
Bill Boarding

- Eye looking down –Z axis, UP = +Y axis
- Compute eye-vector from ModelView:

  \[
  \begin{align*}
  \text{look} &= \text{camera pos} - \text{point pos}; \\
  \text{right} &= \text{up} \times \text{look}; \\
  \text{up} &= \text{look} \times \text{right};
  \end{align*}
  \]

- Rotation about Y:

  \[
  \begin{align*}
  \text{rotation} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
  \text{where:} & \quad \text{rotation} = R \theta
  \end{align*}
  \]

- Build rotation matrix (R) with \( \theta \)
- Transform geometry: \( \text{MR} (\text{Modelview} \times \text{Rotation}) \)

Billboards

- Billboards look = \( \text{camera pos} - \text{point pos} \);
- right = \( \text{up} \times \text{look} \);
- up = \( \text{look} \times \text{right} \);

Billboards Hack

- Trees don't face camera
- Use the Modelview
- Set rotation to identity

```
void billboardCheatSphericalBegin() {
  float modelview[16];
  int i,j;
  // save the current modelview matrix
  glPushMatrix();
  // get the current modelview matrix
  glGetFloatv(GL_MODELVIEW_MATRIX , modelview);
  // undo all rotations
  // beware all scaling is lost as well
  for( i=0; i<3; i++ )
    for( j=0; j<3; j++ ) {
      if ( i==j )
        modelview[i*4+j] = 1.0;
      else
        modelview[i*4+j] = 0.0; }
  // set the modelview with no rotations and scaling
  glLoadMatrixf(modelview);
  // restores the modelview matrix
  glPopMatrix();
}

void billboardEnd() {
  // saves the modelview matrix
  glPushMatrix();
  // get the current modelview matrix
  glGetFloatv(GL_MODELVIEW_MATRIX , modelview);
  // undo all rotations
  if( !m )
    if( !n )
      if( !m )
        m = modelview[0];
      n = modelview[6];
      if( !n )
        n = modelview[12];
    else
      n = modelview[1];
    else
      m = modelview[5];
  else
    m = modelview[2];
  if( !m )
    m = modelview[4];
  if( !n )
    n = modelview[8];
  if( !m )
    m = modelview[14];
  else
    m = modelview[10];
  if( !n )
    n = modelview[16];
  else
    n = modelview[12];
  if( !m )
    m = modelview[18];
  else
    m = modelview[14];

  // saves the modelview matrix
  glPushMatrix();
  // get the current modelview matrix
  glGetFloatv(GL_MODELVIEW_MATRIX , modelview);
  // undo all rotations
  // beware all scaling is lost as well
  for( i=0; i<3; i++ )
    for( j=0; j<3; j++ ) {
      if ( i==j )
        modelview[i*4+j] = 1.0;
      else
        modelview[i*4+j] = 0.0; }
  // set the modelview with no rotations and scaling
  glLoadMatrixf(modelview);
  // restores the modelview matrix
  glPopMatrix();
}
```
Billboards Hack 2

- Trees don’t face camera
- Use the Modelview
- Make billboard cylindrical
- Set part of rotation to identity

Billboards Hack 3

- Modify the vertices of the Billboard pixel
- Reverses the orientations in the Modelview Matrix
- Draw quad using right-up offsets
- Only get modelview once
- Must xform all vertices

How to do cylindrical?

Billboards Correct

- Trees face camera

Billboards Correct

- Trees face camera
- Need
  - Object in world coords
  - Target position (camera) in world coords
- Assume for the object (billboard)
  Right = [1,0,0]
  Up = [0,1,0]
  LookAt = [0,0,1] (which is the normal)

Billboards Correct

objToCamProj is the projection to the XZ plane (set y=0)

1. Normalize objToCamProj
2. \text{aux}=\text{LookAt} \cdot \text{objToCamProj}
3. Up’= \text{lookAt} \times \text{objToCamProj}
4. \text{glRotate}(\text{acos} (\text{aux}), \text{Up’[0]}, \text{Up’[1]}, \text{Up’[2]})
Billboards Correct

1. Normalize objToCamProj
2. aux = LookAt dot objToCamProj
3. Up' = LookAt X objToCamProj
4. glRotate(acos(aux), Up'[0], Up'[1], Up'[2])

Billboards Correct

Object Position in world space

\[ \text{objPosWC} = \text{camPos} + (M_1^{-1})^T \cdot V \]

Billboard Clouds