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
  ```c
  glTexImage2D( GL_TEXTURE_2D, level, …
  ```
- GL mipmap builder routines will build all the textures from a given image
  ```c
  glGenerateMipmap(GL_TEXTURE_2D)
  ```

Anisotropic Filtering

- Anisotropic Filtering
Anisotropic Filtering

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

Light Mapping

- 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

Light Mapping

- 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
Light Mapping

- 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 coordinate transformation 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

Lightmaps

- Adding local light to scene

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
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)

Spotlights as Lightmap

- **Special Case**
  - Mapping Single Spotlight Texture Pattern

  Use texture transformation matrix to perform spotlight texture coordinates transformations.

Lightmaps

- **Creating a lightmap**
  - Light white, tessellated surface with local light
  - Render, capture image as texture
  - Texture contains ambient and diffuse lighting
  - Lighting 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 (just fragment programming)
Lightmaps

- Creating local contribution
  ![Unlit Scene](image1)
  ![Lightmap Intensity](image2)
  ![Local Light Contribution](image3)

- Adding local light to scene
  ![OpenGL Lighting](image4)
  ![Combined Image](image5)

Lightmaps in Quake2

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

Packing Many Lightmaps into a Single Texture

- Quake 2 light map texture image example

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)
    - not a big problem
    - may need to go to multi-pass rendering (fill-bound app)
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 sampler in the fragment program.
- 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:
  `glGetIntegeri64v(GL_MAX_COMBINED_TEXTURE_IMAGE_UNITS)`

OpenGL Multitexturing Quick Tutorial

- Configuring multitextures:
  ```
  GLuint textures[3];
  glGenTextures(3, &textures);
  
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, textures[0]);
  glActiveTexture(GL_TEXTURE1);
  glBindTexture(GL_TEXTURE_2D, textures[1]);
  glActiveTexture(GL_TEXTURE2);
  glBindTexture(GL_TEXTURE_2D, textures[2]);
  
  tex0_uniform_loc = glGetUniformLocation(prog, "tex0");
  glUniform1i(tex0_uniform_loc, 0);
  tex1_uniform_loc = glGetUniformLocation(prog, "tex1");
  glUniform1i(tex1_uniform_loc, 1);
  tex2_uniform_loc = glGetUniformLocation(prog, "tex2");
  glUniform1i(tex2_uniform_loc, 2);
  ```

Texture Units

- Texture units are named GL_TEXTURE0, GL_TEXTURE1, etc.
- The unit names are used with two new functions.
  - `glActiveTexture(texture_unit)` selects the current unit to be affected by texture calls (such as `glBindTexture`, `glTexEnv`, `glTexCoord`).
  - Use vertex attributes to set texture coordinates for each unit.

OpenGL Multitexture Quick Tutorial

- Configuring multitextures:
  ```
  GLuint textures[3];
  glGenTextures(3, &textures);
  
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, textures[0]);
  glActiveTexture(GL_TEXTURE1);
  glBindTexture(GL_TEXTURE_2D, textures[1]);
  glActiveTexture(GL_TEXTURE2);
  glBindTexture(GL_TEXTURE_2D, textures[2]);
  
  layout (binding = 0) uniform sampler tex0;
  layout (binding = 1) uniform sampler tex1;
  layout (binding = 2) uniform sampler tex2;
  ```
OpenGL Multitexture Texture Environments (old way)

• Chain of Texture Environment Stages
  Pre-texturing color
  VertexAttribute(r,g,b)

  tex0  Lookup & filter  GL_MODULATE  #0
  tex1  Lookup & filter  GL_DECAL  #1
  tex2  Lookup & filter  GL_BLEND  #2

Post-texturing color

OpenGL Multitexture Texture Environments (new way)

• Chain of Texture Environment Stages: put it in the shaders!
  varying vec3 lightDir, normal, TexCoord[2];
  varying vec3 lightDir, normal, TexCoord[2];
  varying vec3 lightDir, normal, TexCoord[2];
  void main(){
    TexCoord[0] = TextureMatrix[0] * MultiTexCoord0;
    gl_Position = ftransform();
  }

Detail Texture

Multitexture Lightmapping

Look at SuperBible Example

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.

**Bill Boarding**

- How?

**Billboards**

- Eye looking down –Z axis, UP = +Y axis.
- Compute eye-vector from ModelView:
  
  ```
  \text{look} = \text{camera_pos} - \text{point_pos};
  \text{right} = \text{up} \times \text{look};
  \text{up} = \text{look} \times \text{right};
  ```

- Rotation about Y:

  ```
  \text{rotation_{y}} = \begin{pmatrix}
  1 & 0 & 0 \\
  0 & \cos{\theta} & -\sin{\theta} \\
  0 & \sin{\theta} & \cos{\theta}
  \end{pmatrix}
  ```

  Where:
  
  ```
  \theta = \text{rotation_y} = \frac{\text{up} \cdot \text{right}}{\text{|up|} \text{||right||}}
  ```

- Build rotation matrix (R) with theta.
- Transform geometry: MR (Modelview * Rotation).
Billboards

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

Billboards Hack

- 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 quad
- Reverses the orientations in the Modelview Matrix
- Draw quad using right/up offsets
- Only get modelview once
- Must transform all vertices

\[
a = \text{center} - (\text{right} + \text{up}) \times \text{size}; \\
b = \text{center} + (\text{right} - \text{up}) \times \text{size}; \\
c = \text{center} + (\text{right} + \text{up}) \times \text{size}; \\
d = \text{center} - (\text{right} - \text{up}) \times \text{size};
\]

How to do cylindrical?

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)

\[
\text{objToCamProj is the projection to the XZ plane (set y=0)}
\]

1. Normalize objToCamProj
2. \(\text{aux} = \text{LookAt dot objToCamProj}\)
3. \(\text{Up}' = \text{LookAt X objToCamProj}\)
4. \(\text{glRotate} (\text{acos} (\text{aux}), \text{Up}'[0], \text{Up}'[1], \text{Up}'[2])\)
Billboards Correct

objToCamProj is the projection to the XZ plane (y=0)
1. Normalize objToCamProj
2. aux=LookAt dot objToCamProj
3. Up'= lookAt X objToCamProj
4. glRotate(acos(aux), Up'[0], Up'[1], Up'[2])

Tilt towards Camera
1. Aux'= objToCamProj dot objToCam
2. glRotate(acos(aux'), right[0], right[1], right[2])