Texture Coord Generation

Texture Mapping
Map a 2D Texture onto an Object
– How?
Consider a Cylinder
What if we don’t have a cylinder or sphere?

glTexGen
• Powerful, flexible, underutilized
  – Contour mapping
  – Reflection mapping
  – Lighting effects
  – Atmospheric effects

How TexGen Fits in the T&L Pipe

Reference Plane
• Uses plane equation
  – Ax + By + Cz = D

• Computes dot product
  – coord = Ax + By + Cz + Dw
  – coord is distance from plane

• Computation is “separable”

• Generate texture coordinates from geometry
  – Object space
    • texture is “attached” to object
  – Eye space
    • object moves within texture “field”
  – Sphere map
    • based on reflection vector
Object Linear Mapping

- Texture is “attached” to object
  
  ```
  GLfloat params = {A, B, C, D};
  glTexGenfv(GL_S, GL_OBJECT_PLANE, params);  
  glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);  
  glEnable(GL_TEXTURE_GEN_S);
  ```

- Default mapping is identity
  
  \((s, t, r, q) = (X_0, Y_0, Z_0, W_0)\);

Object Linear Sample

- Texture is “attached” to object

Eye Linear Mapping

- Demo

  ```
  GLfloat params = {A, B, C, D};
  glTexGenfv(GL_S, GL_EYE_PLANE, params);  
  glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);  
  glEnable(GL_TEXTURE_GEN_S);
  ```

- Default mapping is identity
  
  \((s, t, r, q) = (X_e, Y_e, Z_e, W_e)\);

Eye Linear Sample

- Texture is “fixed” in eye space

Eye Linear Mapping

- Demo
Uses

- Projected shadows
- Spotlights
- Aligning screen space textures
  - NPR “paper” background
- Distance functions
  - point light attenuation
  - texture fog

Sphere Mapping

- Based on reflection vector

```c
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glEnable(GL_TEXTURE_GEN_S);
```

- S, T coordinates only

Sphere Mapping

- Spheremap Details

Sphere Map

- Show PDF

Sphere Mapping

- Based on reflection vector

- Environment mapping
- Pre-computed specular lighting (recall lighting lecture)
Textures and Specular

- Getting Highlights Right

GL_SEPARATE_SPECULAR_COLOR

Filtering

“Optimal” case

Minification

Magnification
Pixel Footprint

Pyramid Textures (Mipmapping)

Linear vs. Nearest

Trilinear

Anisotropic
Anisotropic Filtering

Getting Finer Highlight and Lighting Fx

- Run the computation in fragment programs
  - Needs to be local lighting
    - (no radiosity / global illum)
  - Potentially expensive (slow)
- Dynamic scene, lights
  - Play texture tricks
    - Compute the lighting beforehand, store in texture
    - Need to recompute when scene/lights change

Phong Shading

- Sphere mapping approximates Phong Shading
  - Spheremap texture indexed from reflection vector
  - Computation done in eye space
  - Texture looks like reflection from chrome hemisphere
  - Maps all reflections to hemisphere
    - Center of map reflects back to eye (180- degrees)

Why would texturing help?

- Put linear interpolation BEFORE highly non-linear computation (EXP)
  $I_{specular} = k_{light} (\cos \phi)^{\alpha}$

Phong Shading
Phong Shading

- Spheremap Details

Specular Map Texture looks like a highlight on a sphere projected to a plane

Phong Shading

- Phong Shading with Texture
  - Multipass technique:
    - Render object without specular light
    - Redraw object: no lighting, color, surface texture
    - Modulate intensity with highlight texture
    - Combine images with additive blend
  - Can be done in single pass using fragment prog
    - Need additive texenv

Phong Shading

- Creating Highlight Textures
  - Build an environment map
    - Sphere lit by object’s light
    - Relative light position same as object
    - Copy image into highlight texture
  - New texture when light moves relative to viewer
  - Object movement doesn’t require changing texture

Phong Shading

- Compare the Results

Single pass
Phong Lighting, Gouraud Shading
Phong Lighting, Gouraud Shading
Separate specular
Two pass
Phong Lighting, Phong Shading

Blending

- Combine fragments with pixel values that are already in the framebuffer

\[ \hat{C}_r = src \cdot C_r + dst \cdot C_p \]

Phong Shading

- Steps to adding texture highlights

Diffuse Lit, Textured
White, Modulated by Specular Texture
Combined Image (additive blending)
End of Phong Shading

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)

Texture Tricks

Precompute lighting, Fold into texture vs. Surface Texture plus Lightmap
Spotlights as Lightmap Special Case

- Mapping Single Spotlight Texture Pattern

Use texture matrix to perform spotlight texture coordinates transformations.

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

Creating a lightmap
- Render surface lit by local light
- Create a Texture Map from Image

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

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

Creating local contribution
- Unit Scene
- Lightmap Intensity
- Local Light Contribution
Lightmaps

• Adding local light to scene

OpenGL Lighting

Combined Image

Lightmaps in Quake2

lightmaps only
decal only

combined scene

Packing Many Lightmaps into a Single Texture

• Quake 2 lightmap 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-tesselated surfaces
    • Custom lighting
    • Multiple identical lights
    • Static scene lighting

• Lightmap considerations
  – Lightmaps less helpful:
    • Highly tesselated surfaces
    • Directional lights
    • Combine with other surface effects (e.g., bump-mapping)
      – eats a texture access in fragment programs
      – 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, the 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.
- Most current hardware has from 2 to 8 texture units.
- To get the number of units available: glGetIntegerv(GL_MAX_TEXTURE_UNITS)

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, glTexGen).
- glMultiTexCoord2f(texture_unit, s, t)
  - Sets texture coordinates for one unit

Multitexture Lightmapping

Detail Texture

Billboards

look = camera_pos - point_pos;
right = up x look;
up = look x right;
Billboards

up = arbitrary axis
look = camera_pos - point_pos;
right = up x look;
look = right x up;

[r1 u1 l1 px]
[r2 u2 l2 py]
[r3 u3 l3 pz]
[0 0 0 1]

Billboard Clouds

Slides adapted from
• Siggraph 2004 OpenGl course
• U. Virginia Intro to Graphics
• Michael Gold, NVidia
• Cass Everitt, NVidia
• Paul Heckbert, NVidia

Slide Credits

• Michael I. Gold, NVidia
• Cass Everitt, NVidia
• Paul Heckbert, NVidia