VGP352 – Week 7

- Agenda:
  - Nonphotorealistic Rendering
    - Cel shading
    - Gooch technical illustration
  - Post-processing, part 1
    - Texture rectangles
    - Full-screen post-processing overview
    - “Ripple” effect

23-February-2010

Non-photorealistic Rendering (NPR)

- From Wikipedia:

  Non-photorealistic rendering (NPR) is an area of computer graphics that focuses on enabling a wide variety of expressive styles for digital art.

Non-photorealistic Rendering (NPR)

- From Wikipedia:

  > Non-photorealistic rendering (NPR) is an area of computer graphics that focuses on enabling a wide variety of expressive styles for digital art.

- In other words, NPR attempts to exaggerate or use alternate representations of imagery to convey or highlight a particular mood or message
  - Cel shading (a.k.a. “toon” rendering)
  - Painterly rendering
  - Technical illustrations
  - etc.

Cel Shading

- Several common cartoon image styles:
  - Character regions filled with solid, single-tone colors
  - Regions filled with two tones: light and dark
  - Regions filled with three tones: light, dark, and highlight

Cel Shading

- Several common cartoon image styles:
  - Character regions filled with solid, single-tone colors
  - Regions filled with two tones: light and dark
  - Regions filled with three tones: light, dark, and highlight
  - Each is easy to produce on a computer

Cel Shading

- Single tone coloring

23-February-2010

Cel Shading

- Single tone coloring
  - Solid coloring (flat shading) *without* lighting

Cel Shading

- Single tone coloring
  - Solid coloring (flat shading) *without* lighting
- Two-tone coloring

Cel Shading

- Single tone coloring
  - Solid coloring (flat shading) *without* lighting
- Two-tone coloring
  - Driven by surface lighting
  - If lighting is above some threshold, use the lighter color
  - Otherwise use the darker color

Cel Shading

- Single tone coloring
  - Solid coloring (flat shading) *without* lighting
- Two-tone coloring
  - Driven by surface lighting
  - If lighting is above some threshold, use the lighter color
  - Otherwise use the darker color
  - Calculate $\mathbf{n} \cdot \mathbf{l}$ *per vertex* and interpolate across surface, check value per fragment
    - Classically done using texture look-ups, but is faster using conditional assignments on shader hardware

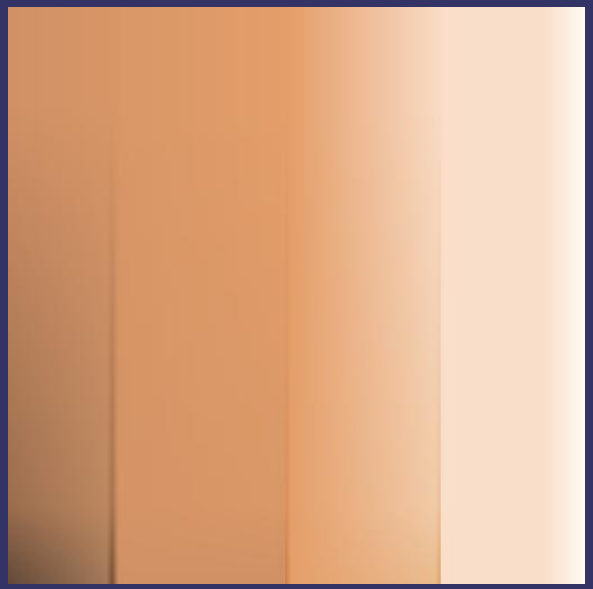
Cel Shading

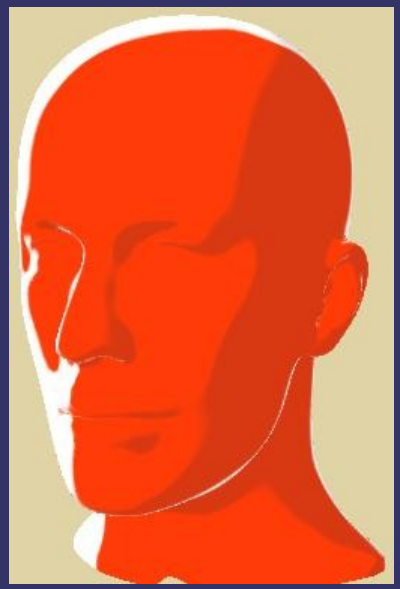
- Shadow-and-highlight method:
  - Calculate $\mathbf{n} \cdot \mathbf{l}$ per vertex
  - Look-up in texture that is half shadow, almost half non-shadow, and a small amount “highlight”
    - Highlight is usually 1 texel

Image from [Lake 2000]

Cel Shading

- X-toon method replaces 1D $\mathbf{n} \cdot \mathbf{l}$ look-up with a 2D $\{\mathbf{n} \cdot \mathbf{l}, (\mathbf{n} \cdot \mathbf{v})^s\}$ look-up
  - Provides a Fresnel-like “rim light” effect





Images from [Barla 2006]

23-February-2010

Cel Shading

- Team Fortress 2 takes this several steps further...

$$\mathbf{k}_{\mathrm{d}}\left[\mathrm{a}(\mathbf{n})+\sum_{\text{all lights}}\mathbf{c}_{\mathrm{i}}\,\mathrm{w}\left(\left(\alpha(\mathbf{n}\cdot\mathbf{l}_i)+\beta\right)^{\gamma}\right)\right]+$$
$$\sum_{\text{all lights}}\left[\mathbf{k}_{\mathrm{s}}\mathbf{c}_{\mathrm{i}}\max\left(f_{\mathrm{s}}(\mathbf{v}\cdot\mathbf{r})^{k_{\text{spec}}},f_{\mathrm{r}}\mathbf{k}_{\mathrm{r}}(\mathbf{v}\cdot\mathbf{r})^{k_{\text{rim}}}\right)\right]+(\mathbf{n}\cdot\mathbf{u})f_{\mathrm{r}}\mathbf{k}_{\mathrm{r}}\mathrm{a}(\mathbf{v})$$

Cel Shading

- Team Fortress 2 takes this several steps further...

$$\mathbf{k}_{\mathrm{d}}\left[\mathrm{a}(\mathbf{n})+\sum_{\text{all lights}}\mathbf{c}_{\mathrm{i}}\,\mathrm{w}\left(\left(\alpha(\mathbf{n}\cdot\mathbf{l}_i)+\beta\right)^{\gamma}\right)\right]+$$

$$\sum_{\text{all lights}}\left[\mathbf{k}_{\mathrm{s}}\mathbf{c}_{\mathrm{i}}\max\left(f_{\mathrm{s}}(\mathbf{v}\cdot\mathbf{r})^{k_{\text{spec}}},f_{\mathrm{r}}\mathbf{k}_{\mathrm{r}}(\mathbf{v}\cdot\mathbf{r})^{k_{\text{rim}}}\right)\right]+(\mathbf{n}\cdot\mathbf{u})f_{\mathrm{r}}\mathbf{k}_{\mathrm{r}}\mathrm{a}(\mathbf{v})$$

Light color
Surface specular color
Surface diffuse color

23-February-2010

Cel Shading

- Team Fortress 2 takes this several steps further...

$$\mathbf{k}_{\mathrm{d}}\left[\mathrm{a}(\mathbf{n})+\sum_{\text{all lights}}\mathbf{c}_{\mathrm{i}}\,\mathrm{w}\left(\left(\alpha(\mathbf{n}\cdot\mathbf{l}_i)+\beta\right)^{\gamma}\right)\right]+$$

$$\sum_{\text{all lights}}\left[\mathbf{k}_{\mathrm{s}}\mathbf{c}_{\mathrm{i}}\max\left(f_{\mathrm{s}}(\mathbf{v}\cdot\mathbf{r})^{k_{\text{spec}}},f_{\mathrm{r}}\mathbf{k}_{\mathrm{r}}(\mathbf{v}\cdot\mathbf{r})^{k_{\text{rim}}}\right)\right]+(\mathbf{n}\cdot\mathbf{u})f_{\mathrm{r}}\mathbf{k}_{\mathrm{r}}\mathrm{a}(\mathbf{v})$$

Directional ambient term

23-February-2010

Cel Shading

- Team Fortress 2 takes this several steps further...

$$\mathbf{k}_{\mathrm{d}}\left[\mathrm{a}(\mathbf{n})+\sum_{\text{all lights}}\mathbf{c}_{\mathrm{i}}\,\mathrm{w}\left(\left(\alpha(\mathbf{n}\cdot\mathbf{l}_i)+\beta\right)^{\gamma}\right)\right]+$$

$$\sum_{\text{all lights}}\left[\mathbf{k}_{\mathrm{s}}\,\mathbf{c}_{\mathrm{i}}\max\left(f_{\mathrm{s}}(\mathbf{v}\cdot\mathbf{r})^{k_{\mathrm{spec}}},f_{\mathrm{r}}\,\mathbf{k}_{\mathrm{r}}(\mathbf{v}\cdot\mathbf{r})^{k_{\mathrm{rim}}}\right)\right]+(\mathbf{n}\cdot\mathbf{u})f_{\mathrm{r}}\,\mathbf{k}_{\mathrm{r}}\,\mathrm{a}(\mathbf{v})$$

"Modified" Lambertian term

Cel Shading

- Team Fortress 2 takes this several steps further...

$$\mathbf{k}_{\mathrm{d}}\left[\mathrm{a}(\mathbf{n})+\sum_{\text{all lights}}\mathbf{c}_{\mathrm{i}}\,\mathrm{w}\left(\left(\alpha(\mathbf{n}\cdot\mathbf{l}_i)+\beta\right)^{\gamma}\right)\right]+$$

$$\sum_{\text{all lights}}\left[\mathbf{k}_{\mathrm{s}}\,\mathbf{c}_{\mathrm{i}}\max\left(f_{\mathrm{s}}(\mathbf{v}\cdot\mathbf{r})^{k_{\mathrm{spec}}},f_{\mathrm{r}}\,\mathbf{k}_{\mathrm{r}}(\mathbf{v}\cdot\mathbf{r})^{k_{\mathrm{rim}}}\right)\right]+(\mathbf{n}\cdot\mathbf{u})f_{\mathrm{r}}\,\mathbf{k}_{\mathrm{r}}\,\mathrm{a}(\mathbf{v})$$

Two specular terms!

Cel Shading

- Team Fortress 2 takes this several steps further...

$$\mathbf{k}_{\mathrm{d}}\left[\mathrm{a}(\mathbf{n})+\sum_{\text{all lights}}\mathbf{c}_{\mathrm{i}}\,\mathrm{w}\left(\left(\alpha(\mathbf{n}\cdot\mathbf{l}_i)+\beta\right)^{\gamma}\right)\right]+$$

$$\sum_{\text{all lights}}\left[\mathbf{k}_{\mathrm{s}}\,\mathbf{c}_{\mathrm{i}}\max\left(f_{\mathrm{s}}(\mathbf{v}\cdot\mathbf{r})^{k_{\mathrm{spec}}},f_{\mathrm{r}}\,\mathbf{k}_{\mathrm{r}}(\mathbf{v}\cdot\mathbf{r})^{k_{\mathrm{rim}}}\right)\right]+(\mathbf{n}\cdot\mathbf{u})\,f_{\mathrm{r}}\,\mathbf{k}_{\mathrm{r}}\,\mathrm{a}(\mathbf{v})$$

Fairly standard specular term

- f_{s} – artist tuned Fresnel term
- k_{spec} – specular from texture or constant

Cel Shading

- Team Fortress 2 takes this several steps further...

$$\mathbf{k}_{\mathrm{d}}\left[\mathrm{a}(\mathbf{n})+\sum_{\text{all lights}}\mathbf{c}_{\mathrm{i}}\,\mathrm{w}\left(\left(\alpha(\mathbf{n}\cdot\mathbf{l}_i)+\beta\right)^{\gamma}\right)\right]+$$

$$\sum_{\text{all lights}}\left[\mathbf{k}_{\mathrm{s}}\mathbf{c}_{\mathrm{i}}\max\left(f_{\mathrm{s}}(\mathbf{v}\cdot\mathbf{r})^{k_{\mathrm{spec}}},f_{\mathrm{r}}\mathbf{k}_{\mathrm{r}}(\mathbf{v}\cdot\mathbf{r})^{k_{\mathrm{rim}}}\right)\right]+(\mathbf{n}\cdot\mathbf{u})f_{\mathrm{r}}\mathbf{k}_{\mathrm{r}}\mathrm{a}(\mathbf{v})$$

Rim specular term

- f_{r} – rim Fresnel $(1-(\mathbf{n}\cdot\mathbf{v}))^4$
- k_{rim} – constant rim exponent

Cel Shading

- Team Fortress 2 takes this several steps further...

$$\mathbf{k}_{\mathrm{d}}\left[\mathrm{a}(\mathbf{n})+\sum_{\text{all lights}}\mathbf{c}_{\mathrm{i}}\,\mathrm{w}\left(\left(\alpha(\mathbf{n}\cdot\mathbf{l}_i)+\beta\right)^{\gamma}\right)\right]+$$

$$\sum_{\text{all lights}}\left[\mathbf{k}_{\mathrm{s}}\,\mathbf{c}_{\mathrm{i}}\max\left(f_{\mathrm{s}}(\mathbf{v}\cdot\mathbf{r})^{k_{\text{spec}}},f_{\mathrm{r}}\,\mathbf{k}_{\mathrm{r}}(\mathbf{v}\cdot\mathbf{r})^{k_{\text{rim}}}\right)\right]+(\mathbf{n}\cdot\mathbf{u})f_{\mathrm{r}}\,\mathbf{k}_{\mathrm{r}}\,\mathrm{a}(\mathbf{v})$$

Extra term to make rim highlights come from above

- $\mathbf{u}$ – up vector

References

Lake, A., Marshall, C., Harris, M., and Blackstein, M. 2000. Stylized rendering techniques for scalable real-time 3D animation. In *Proceedings of the 1st international Symposium on Non-Photorealistic Animation and Rendering* (Annecy, France, June 05 - 07, 2000). NPAR '00. ACM, New York, NY, 13-20. http://www.cs.utah.edu/npr/papers/Lake_NPAR00.pdf

Barla, P., Thollot, J., and Markosian, L. 2006. X-toon: an extended toon shader. In *Proceedings of the 4th international Symposium on Non-Photorealistic Animation and Rendering* (Annecy, France, June 05 - 07, 2006). NPAR '06. ACM, New York, NY, 127-132. http://artis.imag.fr/Publications/2006/BTM06a/x-toon.pdf

Mitchell, J. L., Francke, M., and Eng, D. 2007. Illustrative rendering in Team Fortress 2. In *ACM SIGGRAPH 2007 Courses* (San Diego, California, August 05 - 09, 2007). SIGGRAPH '07. ACM, New York, NY, 19-32. http://www.valvesoftware.com/publications.html

Cel Boundary Inking

- Anyone who has seen a cartoon or a comic book knows that certain boundaries are “inked”

Cel Boundary Inking

- Anyone who has seen a cartoon or a comic book knows that certain boundaries are “inked”
- Four main types of edges need inking:

Cel Boundary Inking

- Anyone who has seen a cartoon or a comic book knows that certain boundaries are “inked”
- Four main types of edges need inking:
  - Border edges – edges not shared by two polygons

Cel Boundary Inking

- Anyone who has seen a cartoon or a comic book knows that certain boundaries are “inked”
- Four main types of edges need inking:
  - Border edges – edges not shared by two polygons
  - Crease edges – edges where the angle between the two surfaces is too sharp
    - This angle is called the *dihedral angle*

Cel Boundary Inking

- Anyone who has seen a cartoon or a comic book knows that certain boundaries are “inked”
- Four main types of edges need inking:
  - Border edges – edges not shared by two polygons
  - Crease edges – edges where the angle between the two surfaces is too sharp
    - This angle is called the *dihedral angle*
  - Material edge – boundary between two different colors or materials

Cel Boundary Inking

- Anyone who has seen a cartoon or a comic book knows that certain boundaries are “inked”
- Four main types of edges need inking:
  - Border edges – edges not shared by two polygons
  - Crease edges – edges where the angle between the two surfaces is too sharp
    - This angle is called the *dihedral angle*
  - Material edge – boundary between two different colors or materials
  - Silhouette edges – edges where one border polygon faces towards the viewer and the other faces away

Cel Boundary Inking

- Most boundary types are calculated during authoring or as a pre-processing step
  - Border edges – edges are added by the artist, by the authoring tool, or are detected in a pre-processing step
  - Crease edges – dihedral angle is calculated during pre-processing. If $\mathbf{n}_{surface1} \cdot \mathbf{n}_{surface2} < \cos(60^\circ)$, the edge is a crease
  - Material edge – handled the same as border edges

Silhouette Edge Rendering

- Silhouette edges are view-dependent and must be calculated at run-time
  - Conceptually similar to drawing fins in shells-and-fins fur rendering
- Several broad classes of implementations:
  - Surface angle
  - Added geometry
  - Image processing
  - Explicit edge detection

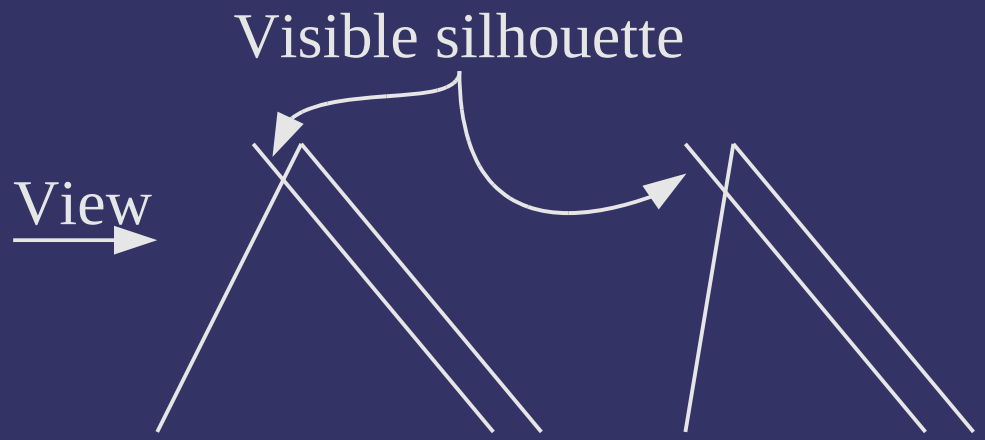
Silhouette Edge Rendering

- Surface angle test is similar to two-tone cel shading
  - Examine angle between $\mathbf{v}$ and $\mathbf{n}$
  - If angle is near 90°, use silhouette color
- Pros / cons:
  - *Really* easy to implement
  - Doesn't work on all models
    - Generally fails on models with large flat surfaces
    - Only worked on about 25% of the models in the game *Cel Damage*[1]

[1] *Real-Time Rendering*, p. 295

Silhouette Edge Rendering

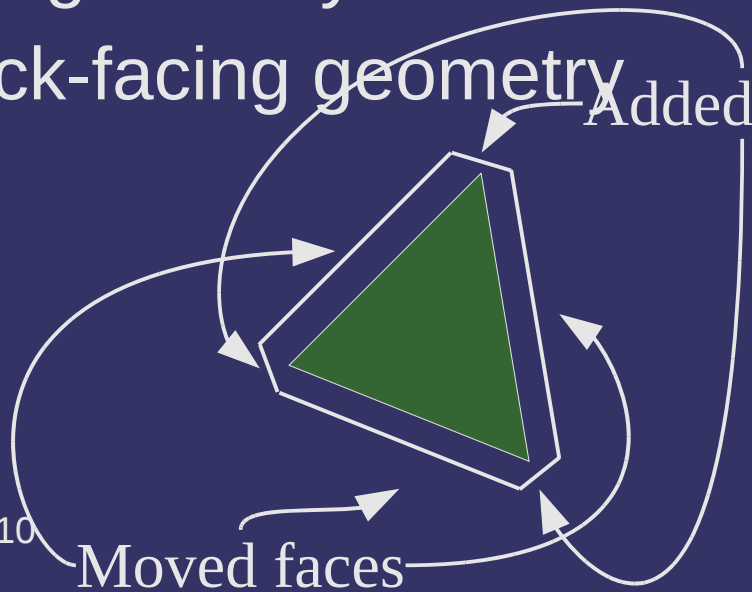
- Back-face biasing:
  - Render back-facing geometry by moving it towards the camera by some small delta



  - Amount to bias back-face depends on both slope of back-face and slope of front-face

Silhouette Edge Rendering

- Edge expansion:
  - Move each face out by some distance along the plane's normal
    - *Not* the vertex normal!
    - Adjust the distance according to the desired silhouette thickness
  - Create new geometry to fill in the gaps
  - Render back-facing geometry



Silhouette Edge Rendering

- Shell expansion:
  - Similar to edge expansion
  - Render shell as object geometry expanded along vertex normals
    - Normals must be identical for vertices shared by two polygons
    - Otherwise degenerate edge polygons must be added
    - Render only back-faces of shell

Silhouette Edge Rendering

- Image processing:
  - Render surface normal and depth a texture
    - Store normal in RGB and most significant portion of depth in alpha
  - Process texture with separable edge detection filter
    - Card and Mitchell recommend using the Sobel edge detection filter
    - Store each pass in a texture
    - Composite both textures together over scene

Silhouette Edge Rendering

- Explicit edge detection:
  - Draw each edge of the object as a line
  - At each vertex, store the normals of the two adjoining polygons
  - If one normal points towards the viewer and the other away, draw the line as a silhouette
  - If the two normals point significantly away from each other, draw the line as a crease

Gooch-style Technical Illustration

- Many similar ideas to cel shading
  - Use alternate shading
  - Highlight creases
  - Highlight silhouettes

Gooch-style Technical Illustration

- Shade objects from warm to cool instead of light to dark
  - Still conveys information about the curvature of the object
  - Maintains visibility of details in areas that would be dark or difficult to light

Gooch-style Technical Illustration

- Shade objects from warm to cool instead of light to dark
  - Still conveys information about the curvature of the object
  - Maintains visibility of details in areas that would be dark or difficult to light
- Shade in similar manner to cel shading
  - Calculate $\mathbf{n} \cdot \mathbf{l}$ per vertex
  - Use interpolated value per fragment to look up in a 1D blue-green to yellow-orange gradient texture

Gooch-style Technical Illustration

- Draw crease edges in white
  - This helps provide information about the model's orientation

Gooch-style Technical Illustration

- Draw crease edges in white
  - This helps provide information about the model's orientation
- Draw silhouette edges in black
  - If an edge is *both* a crease and a silhouette, it should be drawn as a silhouette

Gooch-style Technical Illustration

- Draw crease edges in white
  - This helps provide information about the model's orientation
- Draw silhouette edges in black
  - If an edge is *both* a crease and a silhouette, it should be drawn as a silhouette
- Silhouette and crease edges are handled differently, so the image processing method of inking probably can't be used
  - Using the explicit edge detection method allows silhouettes and creases to be drawn in a single pass

References

Gooch, B., Sloan, P. J., Gooch, A., Shirley, P., and Riesenfeld, R. 1999. Interactive technical illustration. In *Proceedings of the 1999 Symposium on interactive 3D Graphics* (Atlanta, Georgia, United States, April 26 - 29, 1999). I3D '99. ACM, New York, NY, 31-38. http://www.cs.utah.edu/~bgooch/ITI/

Texture Rectangle

- Cousin to 2D textures
  - Interface changes:
    - New texture target: `GL_TEXTURE_RECTANGLE_ARB`
    - New sampler type: `sampler2DRect`, `sampler2DRectShadow`
    - New sampler functions: `texture2DRect`, `texture2DRectProj`, etc.
  - Limitations:
    - No mipmaps
    - Minification filter must be `GL_LINEAR` or `GL_NEAREST`
    - Wrap mode must be one of `GL_CLAMP_TO_EDGE`, `GL_CLAMP_TO_BORDER`, or `GL_CLAMP`

Texture Rectangle

- Added features:
  - Dimensions need not be power of two
    - Alas, now only a “feature” on old hardware
  - Accessed by non-normalized coordinates
    - Coordinates are $[0, w] \times [0, h]$

Post-processing Effects

- Apply an *image space* effect to the rendered scene *after* it has been drawn
  - Examples:
    - Blur
    - Enhance contrast
    - Heat “ripple”
    - Color-space conversion (e.g., black & white, sepia, etc.)
    - Many, *many* more

Post-processing Effects

- Overview:
  - Render scene to off-screen target (framebuffer object)
    - Off-screen target should be same size as on-screen window
    - Additional information may need to be generated
  - Render single, full-screen quad to window
    - Use original off-screen target as source texture
    - Configure texture coordinates to cover entire texture
      - Texture rectangles are *really* useful here
    - Configure fragment shader to perform desired effect

Post-processing Effects

- Configure projection matrix to remap [0, 0] $\times$ [w, h] to [-1, 1] $\times$ [-1, 1] with parallel perspective

$$\begin{bmatrix} \frac{2}{width} & 0 & 0 & -1 \\ 0 & \frac{2}{height} & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - This is the same as the old `glOrtho` function

Post-processing Effects

- Draw two full-screen triangles
  - Use pixel coordinates for both vertex positions and texture coordinates
  - This assumes texture rectangles are being used

Post-processing Effects

- May need to access many neighbor texels in the fragment shader
  - Can calculate these coordinates in the fragment shader, but this uses valuable instructions
  - Instead use all of the available varying slots and pre-calculate offset coordinates in the vertex shader
    - Query `GL_MAX_VARYING_FLOATS` to determine how many slots are available

Post-processing Effects

- Offset texel locations can also be accessed with `textureOffset` and friends

```
vec4 textureOffset(sampler2D s, vec2 p,
    ivec2 offset);
```

  - Integer offset must be known at *compile* time
  - Requires GLSL 1.30.
  - Available with `EXT_gpu_shader4` as `texture2DOffset`, `texture2DRectOffset`, etc.

Ripple Effect

Note the frame-to-frame difference

Image from Enemy Territory: Quake Wars, © Copyright 2007 id Software, Inc.

23-February-2010

Ripple Effect

- Render multiple passes:
  1) Render scene normally to one texture
  2) Render water surface to a separate texture
     - Instead of color, render a perturbation vector
     - Clear color is a perturbation vector of {0, 0}
  3) Render final scene by using water texture to select texels from scene texture
  4) Render water over final scene

Ripple Effect

Note the bleeding of out-of-water elements into the ripples

Image from Enemy Territory: Quake Wars, © Copyright 2007 id Software, Inc.

23-February-2010

Optimization

- Multiple texture look-ups for ever pixel can be expensive

Optimization

- Multiple texture look-ups for ever pixel can be expensive
  - Can render “effect area” to stencil buffer
  - Perform combine step in two passes:
    - First pass just copies areas where stencil is not set
    - Second pass performs effect in areas where stencil is set
  - Can be extended to select multiple screen-space effects using different stencil values

References

Tutorials for several post-processing effects:

http://www.geeks3d.com/20091116/shader-library-2d-shockwave-post-processing-filter-glsl/

Next week...

- More post-processing effects
  - General image filters
  - Separable filters
  - Depth-of-field
  - High dynamic range (HDR) rendering

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of Intel or the Art Institute of Portland.

OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.

Khronos and OpenGL ES are trademarks of the Khronos Group.

Other company, product, and service names may be trademarks or service marks of others.

23-February-2010