Introduction to Virtual Environments - Spring 2004 - Wernert/Arns

Lecture 3.2 – Perceptually-based Development Issues (II):
Textures, Special Effects, & Cameras

Outline

1. Textures
2. Special Effects
3. Cameras & Navigation

0. Resources

http://tecfa.unige.ch/guides/vrml/cosmoplayer/examples/boink/boink.wrl
http://www.web3d.org/WorkingGroups/media/

http://www.siggraph.org/education/materials/HyperGraph/aliasing/alias2a.htm
http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_1.htm
http://www.cgsd.com/papers/gamma.html

1. Texture Mapping

1.1 Overview

- Texture mapping is the process of applying a 2D image to the surface of a polygon
- Reasons:
  - Increase realism (e.g. wood or fabric texture)
  - Reduce polygonal complexity (e.g. brick texture vs. geometrically-modeled bricks, textured "billboard" objects)
  - Supported in hardware on most systems - adds little or no computational load to your environment.
- 3D textures (a.k.a. solid textures) are available in some APIs
1.2 Texture-Mapping Concepts

1.2.1 Texture Space

- $s = x$ dimension of image
- $t = y$ dimension of image
- Some APIs refer to $(u,v)$ instead of $(s,t)$

1.2.2 Texture Coordinates

- $(s, t)$ pair assigned to each vertex of a 3D object.
- Texture coordinate assignment may be:
  - Automatically assigned by API
  - Explicitly set per-vertex by developer
  - Explicitly set via mapping rules by developer

1.2.3 Texture Wrapping

- By default, $(s, t)$ are in the range $[0, 1]$
- If $s$ or $t$ are $> 1$, then the possible behaviors are:
  - Repeat - repeat (cycle) the texture (e.g. $1.7 \rightarrow .7, 6.2 \rightarrow .2$)
  - Clamping - map all values $> 1$ to $1.0$
1.2.4 Texture Transform

- 2D transformation that can modify texture coordinates through:
  - scaling
  - rotation
  - translation

- maintained separately from geometric transformations

1.2.5 Texture/Material blending model

How is texture mapping combined with the reflection model?

- Modulate - multiplies the shaded color by the texture color (the default). If the texture has an alpha component, the alpha value modulates the object's transparency.

- Decal - replaces the shaded color with the texture color. If the texture has an alpha component, the alpha value specifies the texture's transparency, allowing the object's color to show through the texture.
Blend - uses the texture intensity to blend between the shaded color and a specified constant blend color.

1.2.6 Texture Image Components

- Note: "alpha" component = opacity = 1 - transparency
- 1 component (grayscale or intensity) - good in modulate or blend mode
- 2 component (intensity + alpha) - good in modulate or blend mode for billboards or imposters
- 3 component (color - r, g, b) - basic texture mapping - decal or modulate mode
- 4 component (color + alpha - r, g, b, a) - good for billboarding and other special effects (e.g. moving clouds)

1.3 Advanced Texture Methods & Applications

1.3.1 Environment Mapping

- Developer specifies a virtual surrounding shape (usually a cube, sphere, or plane), the object reflects this environment map, making it appear mirror- or chrome-like.
- Texture coordinates are calculated per-frame and per-vertex based on view vector and normal vector
1.3.2 Bump Mapping

- Intensity of the texture map (bump map) is used to perturb the surface normal at each pixel; lighting calculations are performed with the perturbed normal.
- The actual surface is not modified; may lead to noticeable inconsistencies at object silhouettes.

1.3.3 Light Maps

- Lighting for surfaces is precomputed and stored in a texture map.
  - Best lighting effects require significant computation (e.g., Warn's spotlight model with attenuation coupled with high-polygonal count surface.)
- Light map may be pre-blended with surface texture (e.g., brick image) before application to the polygonal surface.

1.3.4 Animated Textures

- Textures retrieved from frames of a movie (e.g., MPEG, AVI, Quicktime format) or from series of image files.
- Requires timing mechanism to control playback.
- MovieTextures supported in VRML97 standard.

1.3.5 Projective Textures
Textures can be "projected" onto surfaces by applying a projection matrix to the texture coordinates.

1.3.6 Procedural Textures

- Textures generated by mathematical functions
- Good for natural textures, including stone, marble, wood, clouds, etc.

**FIGURE 14.46** Constructing marble texture. (a) A spline curve. (b) Unperturbed marble texture.

**FIGURE 14.47** Marble texture. (a) $A = 1$. (b) $A = 3$. (c) $A = 6$.

1.3.7 Solid Textures

- 3D texel space - volume of voxels (imagine a stack of 2D images)
- Vertices require 3D texture coordinate $((s, t, r) \text{ vs. } (s, t))$
- Work nicely with procedurally-generated textures; allow shapes to be "carved" or extracted from the same "block" of texture, generating natural variations in stones, wooded boards, etc.
1.3.8 3D Texture Mapping

- Rendering procedure to utilize with a 3D (solid) texture
  - In OpenGL, we render a number of 2D slices through a 3D volume
    - 2D slices are rendered perpendicular to viewing direction and are blended from back to front

1.4 Technical Issues for Texture Mapping

1.4.1 Texture Coordinate Generation

- explicitly assigned on a per-vertex basis by programmer/modeler
- representative default mappings for shapes
1.4.2 Perspective Correction

The rendering portion of the texture mapping process occurs at the end of the pipeline (in the rasterization step). Given the texture coordinates at each projected vertex, linear interpolation will produce incorrect results:

This effect is especially noticeable on large planar polygons:
- e.g. football field, long brick hallway

Hyperbolic interpolation of texture coordinates is required.
- Fortunately, this is handled correctly by modern hardware and APIs

1.4.3 Minification and Magnification
What to do when the texture pixels (texels) get projected to a very large or very small number of screen pixels?

**Magnification** - choose between nearest texel (pixelated look) or interpolate gradually between nearest (smooth, but blurry look)

**Minification** - Mip-maps - pre-computed lower levels of resolution - combined, they take up exactly 1/3 the space of the original image

These are handled automatically in most high-level APIs, must be set explicitly with OpenGL

---

2. Special Effects

2.1 Level-of-Detail (LOD) or multiple resolutions

- specify multiple representations of the object and the range of distances over which they should be used

- [http://www.sdsc.edu/~nadeau/Courses/Siggraph98vrml/vrml97/examples/prox1.wrl](http://www.sdsc.edu/~nadeau/Courses/Siggraph98vrml/vrml97/examples/prox1.wrl)
- don't waste effort drawing detailed objects that only project to a few pixels
- LOD models can be generate by hand, with decimation tools, or via API (e.g. OpenGL Optimizer)
- to avoid popping artifacts, advanced APIs such as Performer will perform alpha cross-fade

---

2.2 Alpha Mapping (textures with alpha channel)

- simple polygons (rectangles) with alpha-channel textures on them
  a.k.a. “imposter geometry”
- Need an image file that supports alpha (transparency):
  - PNG - 8 bit alpha
  - GIF - 1 bit alpha
- Need a good image-processing program (e.g. GIMP)
- Example (from Alice3D.org)
2.3 background color and images

- select appropriate background/clearing color
- APIs may support additional features such as
  - ground planes
  - interpolated sky colors
  - background images
- environment map example

2.4 Particle Systems
2.5 Billboards

- geometry (often texture-mapped imposters) that is automatically oriented in world
  - screen-aligned - always parallel to the screen
  - world-aligned - rotates based on some world constraint (usually an arbitrary axis)
- full-screen billboards can provide simple effects - e.g. night vision goggles, scene flashes
  - http://www.sdsc.edu/~nadeau/Courses/Siggraph98vrml/vrml97/examples/robobill.wrl

2.6 Sprites

- may be thought of as screen-aligned billboards with animated, alpha textures
  - originally rendered in separate bit planes
  - may provide additional efficiencies (e.g., limited scene redraw, animation by texture coordinate reassignment -> texture caching, etc.)

2.7 planar reflections and shadows

- geometry may be easily projected onto a flat ground plane
- geometry may be literally mirrored below/behind a high-gloss surface to simulate mirror effects

3. Cameras & Navigation
3.1 Camera Concepts

- **Virtual camera model** - defines viewpoint (position and orientation) as well as projection properties (field of view (lens settings), aspect ratio (film size), etc.)

- **Projection methods**
  - Perspective projection - simulates human eye and camera lens - objects further away appear smaller in size (foreshortening)
  - Orthographic projection - no distortion for distance; preserves parallel lines; useful for design work, not generally for VR

- **Field of view (FOV)** - defines properties of virtual lens, measured either in horizontal (x) or vertical (y) direction
  - Large FOV = wide-angle lens
  - Small FOV = telephoto lens
  - For VR, we want to compute FOV based on relationship between user’s head and the display screen

- **Clipping planes** - define boundaries of the viewable region; provide usability benefits; help to maximize accuracy and efficiency of z-buffer and rendering pipeline
  - Near clipping plane - removes items between near clipping plane and viewpoint; keeps items from obscuring view of other parts of scene; important to keep items from getting too close in VR (minimum of 1 ft)
  - Far clipping plane - removes items beyond some maximum distance (would often project to a very few pixels anyway)

- **Viewport** - rectangular area of the screen where image is mapped; measured in pixels; by default, is the same size as the window. (Note: in general, a window may have multiple viewports)

- **Aspect ratio** - ratio of width to height of the viewport; common aspect ratios include
  - 1:1 - square
  - 4:3 - video, golden ratio
  - 5:4 - 1280x1024
  - 16:9 - HDTV

  Note: most VR APIs will hide most or all of these details from developer

3.2 Generating Stereo Images

- **Toe-in** (quick & dirty, incorrect) method
- just rotate model a few degrees for left and right eyes

- **Off-Axis Projection** (correct) method

  ![Diagram of Off-Axis Projection](image)

  - in order to get correct stereo, we must compute distances to
    - left, right, top, bottom, near, far edges of the view frustum
    - for both left and right eye
    - based on head position and orientation, eye separation, and screen location
    - for each frame

- Stereo Pair samples - must watch to see if they are cross-eyed or wall-eyed
  - cross-eyed example

  ![Cross-eyed Example](image)

  - wall-eyed example (note how sharp and straight edges give stronger stereo effect)

  ![Wall-eyed Example](image)

- real image example (wall-eyed)

  ![Real Image Example](image)
Putting it all together

Once again, most VR APIs and systems will handle and hide these details.

### 3.3 Camera Control or Navigation

- **camera control model** - conceptual model for how the user moves the viewpoint (camera) through the world; common methods include:
  - examiner (world in hand)
  - walk
  - fly
  - helicopter (fly with fixed up orientation)
  - plane (restrict motion to standard coordinate plane)

- Specific methods may be explicitly supported by higher-level APIs or packages
  - e.g., VRML97 recommends a minimum of EXAMINE, WALK, and FLY modes (although the particular interface (mouse) mappings vary greatly between browsers)
  - other considerations include avatar height and step size for collision calculations in the environment
    - [http://www.sdsc.edu/~nadeau/Courses/Siggraph98vrml/vrml97/examples/playyard.wrl](http://www.sdsc.edu/~nadeau/Courses/Siggraph98vrml/vrml97/examples/playyard.wrl)