Texture Mapping and Sampling

CPSC 314

The Rendering Pipeline

Geometry Database

Model/View Transform.

Lighting

Perspective Transform.

Clipping

Scan Conversion

Texturing

Depth Test

Blending

Frame-Buffer

Geometry Processing

Rastenization

Fragment Processing

Texture Mapping

- Real life objects have nonuniform colors, normals
- To generate realistic objects, reproduce coloring & normal variations = texture
- Can often replace complex geometric details

Texture Mapping

Introduced to increase realism
- Lighting/shading models not enough

Hide geometric simplicity
- Images convey illusion of geometry
- Map a brick wall texture on a flat polygon
- Create bumpy effect on surface

Associate 2D information with 3D surface
- Point on surface corresponds to a point in texture
- “Paint” image onto polygon
Color Texture Mapping

Define color (RGB) for each pt on surface

Two approaches
• Surface texture map
• Volumetric texture

Texture Coordinates

Texture image: 2D array of color values (texels)

Assigning texture coordinates (s,t) at vertex with object coordinates (x,y,z,w)

• Use interpolated (s,t) for texel lookup at each pixel
• Use value to modify a polygon’s color
  – Or other surface property
• Specified by programmer or artist
  
  \[ s, t \in [0, 1] \]
  
  \[ \text{glTexCoord2f}(s, t) \]
  
  \[ \text{glVertexf}(x, y, z) \]

Texture Mapping

Textures of other dimensions

• 1D: represent isovalues
  – e.g.: contour lines, temp, ...
  
  \[ \text{glTexCoord1f}(s) \]

Texture Mapping Example

For color-coded height field, use \( s = a z + b \)
Example Texture Map

```
<table>
<thead>
<tr>
<th>(0, 0)</th>
<th>(0, 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 0)</td>
<td>(1, 1)</td>
</tr>
</tbody>
</table>
```

GlTexImage2d(0, 0);
GlVertex3D(0, -2, -2);

GlTexImage2d(0, 0);
GlVertex3D(0, 2, 2);

Fractional Texture Coordinates

```
<table>
<thead>
<tr>
<th>(0, 0)</th>
<th>(.25, 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(.25, 0)</td>
<td>(.5, .5)</td>
</tr>
<tr>
<td>(.5, 0)</td>
<td>(1, 1)</td>
</tr>
</tbody>
</table>
```

Texture Lookup: Tiling and Clamping

**What if s or t is outside the interval [0…1]?**

**Multiple choices**

- Use fractional part of texture coordinates
  - Cyclic repetition
    ```
    glTexParameteri(..., GL_TEXTURE_WRAP_S, GL_REPEAT, GL_TEXTURE_WRAP_T, GL_REPEAT, ...)
    ```
  - Clamp every component to range [0…1]
    ```
    glTexParameteri(..., GL_TEXTURE_WRAP_S, GL_CLAMP, GL_TEXTURE_WRAP_T, GL_CLAMP, ...)
    ```

Tiled Texture Map

```
<table>
<thead>
<tr>
<th>(0, 0)</th>
<th>(0, 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0)</td>
<td>(0, 1)</td>
</tr>
</tbody>
</table>
```

GlTexImage2d(1, 1);
GlVertex3D(x, y, z);

GlTexImage2d(4, 4);
GlVertex3D(x, y, z);
Texture Coordinate Transformation

**Motivation**
- Change scale, orientation of texture on an object

**Approach**
- Texture matrix stack
- Transforms specified (or generated) tex coords
  ```
  glMatrixMode(GL_TEXTURE);
  glLoadIdentity();
  glRotate();
  ...
  ```
- More flexible than changing (s,t) coordinates

Texture Functions

**Ways of applying texture colour:**
- **GL_REPLACE**
  - Directly use as surface color; no lighting effects
- **GL_MODULATE**
  - Modulate surface color: multiply old color by new value
  - Texturing happens after lighting, not relit
- **GL_DECAL**
  - Like replace, but modulate alpha to support transparency
- **GL_BLEND**
  - Blend surface color with existing on-screen colour

```
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, <mode>)
```

Texture Pipeline

<table>
<thead>
<tr>
<th>Parameter space</th>
<th>Transformed parameter space</th>
<th>Texel space</th>
<th>Texel color</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0.32, 0.29)</td>
<td>(0.52, 0.49)</td>
<td>(81, 74)</td>
<td>(0.9, 0.8, 0.7)</td>
</tr>
<tr>
<td>(s, t)</td>
<td>(s', t')</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Texture Objects and Binding

**Texture object**
- An OpenGL data type that keeps textures resident in memory and provides identifiers to easily access them
- Provides efficiency gains over having to repeatedly load and reload a texture
- You can prioritize textures to keep in memory
- OpenGL uses least recently used (LRU) if no priority is assigned

**Texture binding**
- Which texture to use right now
- Switch between preloaded textures
Basic OpenGL Texturing

Create a texture object and fill it with texture data:
- `glGenTextures(num, &indices)` to get identifiers for the objects
- `glBindTexture(GL_TEXTURE_2D, identifier)` to bind
  - Following texture commands refer to the bound texture
- `glTexParameteri(GL_TEXTURE_2D, …, …)` to specify parameters for use when applying the texture
- `glTexImage2D(GL_TEXTURE_2D, …)` to specify the texture data (the image itself)

Enable texturing:
- `glEnable(GL_TEXTURE_2D)`

State how the texture will be used:
- `glTexEnvf(…)`

Specify texture coordinates for the polygon:
- Use `glTexCoord2f(s, t)` before each vertex:
  - `glTexCoord2f(0, 0); glVertex3f(x,y,z);`

Low-Level Details

Large range of functions for controlling layout of texture data
- State how the data in your image is arranged
  - e.g.: `glPixelStorei(GL_UNPACK_ALIGNMENT, 1)` tells OpenGL not to skip bytes at the end of a row
- You must state how you want the texture to be put in memory: how many bits per “pixel”, which channels,…

Textures must have a size of power of 2
- Common sizes are 32x32, 64x64, 256x256
- But don’t need to be square, i.e. 32x64 is fine
- Smaller uses less memory, and there is a finite amount of texture memory on graphics cards

Texture Mapping

Texture coordinate interpolation
- Perspective foreshortening problem
Interpolation: Screen vs. World Space

Screen space interpolation incorrect
- Problem ignored with shading, but artifacts more visible with texturing

![Diagram of screen space interpolation](image)

Texture Coordinate Interpolation

Perspective correct interpolation
- $\alpha$, $\beta$, $\gamma$:
  - Barycentric coordinates of a point $P$ in a triangle
- $s_0$, $s_1$, $s_2$:
  - Texture coordinates of vertices
- $w_0$, $w_1$, $w_2$:
  - Homogeneous coordinates of vertices

\[
\begin{align*}
\alpha \cdot s_0 & / w_0 + \beta \cdot s_1 / w_1 + \gamma \cdot s_2 / w_2 \\
\alpha / w_0 + \beta / w_1 + \gamma / w_2
\end{align*}
\]

Reconstruction

- How to deal with:
  - Pixels that are much larger than texels?
    - Apply filtering, “averaging”
  - Pixels that are much smaller than texels?
    - Interpolate

(image courtesy of Kiriakos Kutulakos, U Rochester)
Interpolating Textures

- Nearest neighbor
- Bilinear
- Hermite

MIPmapping

Use "image pyramid" to precompute averaged versions of the texture.

store whole pyramid in single block of memory

\[ \text{Mem usage} = \frac{4}{3} \text{ of original} \]

MIPmaps

*Multum in parvo -- many things in a small place*

- Prespecify a series of prefiltered texture maps of decreasing resolutions
- Requires more texture storage
- Avoid shimmering and flashing as objects move

`gluBuild2DMipmaps`

- Automatically constructs a family of textures from original texture size down to 1x1

MIPmap storage

*only 1/3 more space required*
Texture Parameters

In addition to color can control other material/object properties
- Surface normal (bump mapping)
- Reflected color (environment mapping)

Bump Mapping: Normals As Texture

Object surface often not smooth – to recreate correctly need complex geometry model
Can control shape “effect” by locally perturbing surface normal
- Random perturbation
- Directional change over region

Bump Mapping

Original surface

A bump map

Lengthening or shortening \(O(u)\) using \(B(u)\)

The vectors to the ‘new’ surface
Displacement Mapping

**Bump mapping gets silhouettes wrong**
- Shadows wrong too

**Change surface geometry instead**
- Need to subdivide surface

**GPU support**
- Bump and displacement mapping not directly supported: require per-pixel lighting
- However: modern GPUs allow for programming both yourself

Environment Mapping

**Cheap way to achieve reflective effect**
- Generate image of surrounding
- Map to object as texture

Sphere Mapping

**Texture is distorted fish-eye view**
- Point camera at mirrored sphere
- Spherical texture mapping creates texture coordinates that correctly index into this texture map

Cube Mapping

**6 planar textures, sides of cube**
- Point camera in 6 different directions, facing out from origin
Cube Mapping

Direction of reflection vector $r$ selects the face of the cube to be indexed

- Co-ordinate with largest magnitude
  
  - e.g., the vector $(-0.2, 0.5, -0.84)$ selects the $-Z$ face

- Remaining two coordinates (normalized by the 3rd coordinate)
  selects the pixel from the face.

  - E.g., $(-0.2, 0.5)$ gets mapped to $(0.38, 0.80)$.

Difficulty in interpolating across faces

Volumetric Texture

Define texture pattern over 3D domain - 3D space containing the object

- Texture function can be digitized or procedural
- For each point on object compute texture from point location in space

Common for natural material/irregular textures (stone, wood, etc...)

Volumetric Bump Mapping

Marble

Bump
Procedural Textures

Generate "image" on the fly, instead of loading from disk

- Often saves space
- Allows arbitrary level of detail

Several good explanations

- Text book Section 10.1
- http://www.noisemachine.com/talk1
- http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

Samples

- Most things in the real world are continuous
- Everything in a computer is discrete
- The process of mapping a continuous function to a discrete one is called sampling
- The process of mapping a discrete function to a continuous one is called reconstruction
- The process of mapping a continuous variable to a discrete one is called quantization
- Rendering an image requires sampling and quantization
- Displaying an image involves reconstruction
Line Segments

- quantized pixel values to 0 or 1
- or, quantize to many shades

Unweighted Area Sampling

Shade pixels wrt area covered by thickened line
Equal areas cause equal intensity, regardless of distance from pixel center to area

- Rough approximation formulated by dividing each pixel into a finer grid of pixels
- Primitive cannot affect intensity of pixel if it does not intersect the pixel

Weighted Area Sampling

Intuitively, pixel cut through the center should be more heavily weighted than one cut along corner

Weighting function, \( W(x,y) \)
- Specifies the contribution of primitive passing through the point \((x, y)\) from pixel center

Images

An image is a 2D function \( I(x, y) \)
- Specifies intensity for each point \((x, y)\)
- (we consider each color channel independently)
Image Sampling and Reconstruction

Convert continuous image to discrete set of samples

Display hardware reconstructs samples into continuous image

- Finite sized source of light for each pixel

Point Sampling an Image

- Simplest sampling is on a grid
- Sample depends solely on value at grid points

Point Sampling

Multiply sample grid by image intensity to obtain a discrete set of points, or samples.

Sampling Errors

Some objects missed entirely, others poorly sampled
- Could try unweighted or weighted area sampling
- But how can we be sure we show everything?

Need to think about entire class of solutions!
Image As Signal

**Image as spatial signal**

2D raster image
- Discrete sampling of 2D spatial signal

1D slice of raster image
- Discrete sampling of 1D spatial signal

---

Sampling Theory

**How would we generate a signal like this out of simple building blocks?**

**Theorem**
- Any signal can be represented as an (infinite) sum of sine waves at different frequencies

---

Sampling Theory in a Nutshell

**Terminology**
- Wavelength – length of repeated sequence on infinite signal
- Frequency – 1/wavelength (number of repeated sequences in unit length)

**Example – sine wave**
- Wavelength = \(2\pi\)
- Frequency = \(1/2\pi\)

---

Summing Waves I

\[
\sin(t) \quad + \quad \sin(3t) \quad + \quad \frac{\sin(5t)}{5} \quad + \quad \frac{\sin(7t)}{7} \quad + \quad \frac{\sin(9t)}{9}
\]
Summing Waves II
represent spatial signal as sum of sine waves (varying frequency and phase shift) very commonly used to represent sound “spectrum”.
1D Sampling and Reconstruction

Problems
- Jaggies – abrupt changes
- Lose data

1D Sampling and Reconstruction

Problems
- Jaggies – abrupt changes

Sampling Theorem
- Continuous signal can be completely recovered from its samples

Iff
- Sampling rate greater than twice highest frequency present in signal

- Claude Shannon
**Nyquist Rate**

*Lower bound on sampling rate*
- Twice the highest frequency component in the image’s spectrum

---

**Falling Below Nyquist Rate**

*When sampling below Nyquist Rate, resulting signal looks like a lower-frequency one*
- This is aliasing!

---

**Aliasing**

*Incorrect appearance of high frequencies as low frequencies*

**To avoid: anti-aliasing**
- Supersample
  - Sample at higher frequency
- Low pass filtering
  - Remove high frequency function parts
  - Aka prefiltering, band-limiting
Supersampling

Low-Pass Filtering

Low-Pass Filtering

Previous Antialiasing Example

Texture mipmapping: low pass filter