Texture Mapping

Jason Lawrence

CS 4810: Graphics

Acknowledgment: slides by Misha Kazhdan, Allison Klein, Tom Funkhouser, Adam Finkelstein and David Dobkin
Textures

We know how to go from this... to this

J. Birn
Textures

But what about this…

...to this?

J. Birn
Textures

• How can we go about drawing surfaces with complex detail?
Textures

• How can we go about drawing surfaces with complex detail?

• We could tessellate the sphere in a complex fashion and then associate the appropriate material properties to each vertex.
Textures

- How can we go about drawing surfaces with complex detail?

- We could use a simple tessellation and use the location of surface points to look up the appropriate color values.
Textures

• Advantages:
  o The 3D model remains simple
  o It is easier to design/modify a texture image than it is to design/modify a surface in 3D.
Textures

Properties:

- Alter shading of individual pixels
- Implemented as part of shading process
- Rely on maps being stored as 1D, 2D, or 3D images
- Subject to aliasing errors
Textures

General Implementation Approach:

• Associate a collection of coordinates \((s_1, \ldots, s_n)\) to every vertex \((0 \leq s_i \leq 1)\)

• Use the color of the image at position \((s_1, \ldots, s_n)\) to define the color of a vertex
Another Example: Brick Wall
Another Example: Brick Wall

+
2D Texture

- Coordinates described by variables $s$ and $t$ and range over interval $(0,1)$
- Texture elements are called *texels*
- Often 4 bytes (rgba) per texel
2D Texture

```c
glBegin(GL_TRIANGLE);
    glTexCoord2f(0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);
    glTexCoord2f(1.0, 0.0);
    glVertex3f(1.0, 0.0, 0.0);
    glTexCoord2f(1.0, 1.0);
    glVertex3f(1.0, 1.0, 0.0);
glEnd();
```
3D Rendering Pipeline (for direct illumination)

- 3D Primitives
  - 3D Modeling Coordinates
  - Modeling Transformation
    - 3D World Coordinates
    - Camera Transformation
      - 3D World Coordinates
      - Lighting
        - 3D Camera Coordinates
        - Projection Transformation
          - 2D Screen Coordinates
          - Clipping
            - 2D Screen Coordinates
            - Viewport Transformation
              - 2D Screen Coordinates
              - Scan Conversion
                - 2D Image Coordinates
                - Image

Texture mapping
Overview

• Texture mapping methods
  o Parameterization
  o Mapping
  o Filtering

• Texture mapping applications
  o Modulation textures
  o Illumination mapping
  o Bump mapping
  o Environment mapping
  o Shadow maps
  o Volume Textures
Parameterization

geometry + image = texture map

• Q: How do we decide *where* on the geometry each color from the image should go?
Option: Unfold/Map Entire Surface

[Piponi2000]
Option: Unfold/Map Entire Surface

- Tricky, because mapped surface may have severe distortions
- However, because texture is continuous, may be easier to think about

Gu et al. 2003
Option: Unfold/Map Entire Surface

- Tricky, because mapped surface may have severe distortions
- However, because texture is continuous, may be easier to think about

In general, it is impossible to parameterize a complex shape to a simple base domain so that both angles and areas are preserved
Option: Atlas

charts  atlas  surface

[Sander2001]
Option: Atlas

- Less distortion on each little piece of atlas
- Need to pack patches to reduce wasted space in texture image
- May be more difficult to think about the relationships between the different pieces
Overview

- Texture mapping methods
  - Parameterization
  - Mapping
  - Filtering

- Texture mapping applications
  - Modulation textures
  - Illumination mapping
  - Bump mapping
  - Environment mapping
  - Shadow Maps
  - Volume textures
Texture Mapping

- Steps:
  - Define texture
  - Specify mapping from surface to texture
  - Lookup texture values during scan conversion
Texture Mapping

- Scan conversion:
  - Interpolate texture coordinates down/across scan lines
  - Do perspective divide at each pixel based on mapping from screen space to 3-space

\[
(s, t) = \alpha (s_1, t_1) + \beta (s_2, t_2) + \gamma (s_3, t_3)
\]
Texture Mapping

Linear interpolation of texture coordinates in screen space

Correct interpolation with perspective divide

Hill Figure 8.42
Overview

• Texture mapping methods
  o Parameterization
  o Mapping
  o Filtering

• Texture mapping applications
  o Modulation textures
  o Illumination mapping
  o Bump mapping
  o Environment mapping
  o Shadow maps
  o Volume Textures
Texture Filtering

Must sample texture to determine color at each pixel in image
Texture Filtering

Must sample texture to determine color at each pixel in image

• In general, the transformation from screen space to texture space does not preserve area
Texture Filtering

Must sample texture to determine color at each pixel in image

• In general, the transformation from screen space to texture space does not preserve area

• Need to compute the average of the pixels in texture space to get the color for screen space

Minification
Texture Filtering

Must sample texture to determine color at each pixel in image

- In general, the transformation from screen space to texture space does not preserve area
- Need to compute the average of the pixels in texture space to get the color for screen space

If the distortion is very large, this will require a lot of texture look-ups/adds.
Texture Filtering

Size of filter depends on the projective deformation

- Can prefilter images for better performance
  - Mip maps
  - Summed area tables

Minification
Mip Maps

- Keep textures prefiltered at multiple resolutions
  - For each pixel, use the mip-map closest level
  - Fast, easy for hardware
Mip Maps

- Keep textures prefiltered at multiple resolutions
  - For each pixel, use the mip-map closest level
  - Fast, easy for hardware
Mip Maps

- Keep textures prefILTERed at multiple resolutions
  - For each pixel, use the mip-map closest level
  - Fast, easy for hardware

Average over many pixels
Mip Maps

- Keep textures prefiltered at multiple resolutions
  - For each pixel, use the mip-map closest level
  - Fast, easy for hardware

Again: we’re trading aliasing for blurring!
Mip Maps

- Keep textures prefiltered at multiple resolutions
  - For each pixel, use the mip-map closest level
  - Fast, easy for hardware

- This type of filtering is isotropic:
  - It doesn’t take into account that there is more compression in the vertical direction than in the horizontal one

Again: we’re trading aliasing for blurring!
Summed-area tables

Key Idea:

- Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.

\[ \text{Sum}([a, b] \times [c, d]) = \int \int f(x, y) \, dy \, dx \]
Summed-area tables

Key Idea:

• Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.

• Perform the integration quickly by pre-computing integrals and leveraging the formula

\[
\int_0^b \int_0^d f(x, y) \, dy \, dx = \int_0^b \int_0^c f(x, y) \, dy \, dx - \int_0^b \int_0^c f(x, y) \, dy \, dx - \int_0^a \int_0^c f(x, y) \, dy \, dx + \int_0^a \int_0^d f(x, y) \, dy \, dx
\]
Summed-area tables

Key Idea:

• Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.

• Perform the integration quickly by pre-computing integrals and leveraging the formula

\[
\int_{a}^{b} \int_{c}^{d} f(x, y) \, dy \, dx = \int_{a}^{b} \int_{c}^{d} f(x, y) \, dy \, dx - \int_{0}^{b} \int_{0}^{a} f(x, y) \, dy \, dx - \int_{0}^{d} \int_{0}^{c} f(x, y) \, dy \, dx + \int_{0}^{d} \int_{0}^{b} f(x, y) \, dy \, dx
\]
### Summed-area tables

**Key Idea:**

- Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.

- Perform the integration quickly by pre-computing integrals and leveraging the formula:

\[
\int\int f(x, y) dy dx = \int f(x) dx \left[ \int f(y) dy \right]_{a}^{b} - \int f(x) dx \left[ \int f(y) dy \right]_{c}^{d} - \int f(x) dx \left[ \int f(y) dy \right]_{0}^{t} + \int f(x) dx \left[ \int f(y) dy \right]_{0}^{0}
\]
Summed-area tables

Key Idea:

• Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.

• Perform the integration quickly by pre-computing integrals and leveraging the formula:

\[
\int \int f(x,y) \, dy \, dx = \int \int f(x,y) \, dy \, dx - \int \int f(x,y) \, dy \, dx - \int \int f(x,y) \, dy \, dx + \int \int f(x,y) \, dy \, dx
\]
Summed-area tables

Key Idea:

• Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.

• Perform the integration quickly by pre-computing integrals and leveraging the formula

\[
\int \int f(x,y)dydx = \int \int f(x,y)dydx - \int \int f(x,y)dydx - \int \int f(x,y)dydx + \int \int f(x,y)dydx
\]
Summed-area tables

Key Idea:

• Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.

• Perform the integration quickly by pre-computing integrals and leveraging the formula

$$\int \int f(x, y) dy dx = \int \int f(x, y) dy dx - \int \int f(x, y) dy dx - \int \int f(x, y) dy dx + \int \int f(x, y) dy dx$$
Summed-area tables

- Precompute the values of the integral:
  \[ S(a, b) = \int_0^a \int_0^b f(x, y) \, dy \, dx \]

- Each texel is the sum of all texels below and to the left of it

Original image

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Summed area table

<table>
<thead>
<tr>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Summed-area tables

• Now, suppose I have some pixel *on screen* that maps to these pixels in my texture. What to do?
  - Explicitly computing the average (applying a box filter) is too slow!

Original image

1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
Summed-area tables

• Now, suppose I have some pixel on screen that maps to these pixels in my texture. What to do?
  - Explicitly computing the average (applying a box filter) is too slow!
  - Use summed-area table formula
    \[
    \text{Sum}([0,1] \times [3,3]) = S(3,3) - S(0,3) - S(3,1) + S(0,1)
    \]
    \[
    = 16 - 8 - 4 + 2 = 6
    \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Original image

<table>
<thead>
<tr>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Summed-area table
Summed-area tables

- Now, suppose I have some pixel *on screen* that maps to these pixels in my texture. What to do?
  - Explicitly computing the average (applying a box filter) is too slow!
  - Use summed-area table formula
    \[ \text{Sum}([0,1] \times [3,3]) = S(3,3) - S(0,3) - S(3,1) + S(0,1) \]
    \[ = 16 - 8 - 4 + 2 = 6 \]
    \[ \text{Average}([0,1] \times [3,3]) = \frac{\text{Sum}([0,1] \times [3,3])}{\text{Area}([0,1] \times [3,3])} = \frac{6}{6} = 1 \]
Overview

• Texture mapping methods
  o Parameterization
  o Mapping
  o Filtering

• Texture mapping applications
  o Modulation textures
  o Illumination mapping
  o Bump mapping
  o Environment mapping
  o Volume Textures
Modulation textures

Map texture values to scale factor

\[ I = T(s,t)(I_E + K_A I_A + \sum L (K_D (N \cdot L) + K_S (V \cdot R)^n) S_L I_L + K_T I_T + K_S I_S) \]
Illumination Mapping

Map texture values to any material parameter

Modulation

Diffuse

\[ I = I_E + K_A I_A + \sum_L \left[ T(s,t)(N \cdot L) + K_S (V \cdot R)^n \right] S_L I_L + K_T I_T + K_S I_S \]
Illumination Mapping

Map texture values to any material parameter

Modulation

Diffuse

Specular

$$I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + (s,t)(V \cdot R)^n) s_L I_L + K_T I_T + K_S I_S$$
Bump Mapping

- Recall that many parts of our lighting calculation depend on surface normals

\[ I = I_E + K_A I_A + \sum_L \left( K_D (N \cdot L) + K_S (V \cdot R)^n \right) B_L I_L + K_T I_T + K_S I_S \]
Bump Mapping

\[ n_0 \rightarrow n_1 \]

P. Rheingans
Bump Mapping

Phong shading approximates smoothly curved surface

P. Rheingans
Bump Mapping

Phong shading approximates smoothly curved surface

We can store perturbations to normals in a texture map

P. Rheingans
Bump Mapping

Phong shading approximates smoothly curved surface

Now Phong shading gives the appearance of a bumpy surface
Bump Mapping

H&B Figure 14.100
Bump Mapping

Note that bump mapping does not change object silhouette
Environment Mapping

- Generate a spherical/cubic map of the environment around the model.
Environment Mapping

• Generate a spherical/cubic map of the environment around the model.

• Texture values are reflected off surface patch
Environment Mapping

Texture values are reflected off surface patch

P. Debevec
Environment Maps / Light Probes
Cube Maps
Solid textures

Texture values indexed by 3D location \((x,y,z)\)

- Expensive storage, or
- Compute on the fly, e.g. Perlin noise