Texture Mapping

Michael Kazhdan

(601.457/467)

HB Ch. 14.8,14.9
FvDFH Ch. 16.3, 16.4.5, 16.6
Textures

We know how to go from this… to this.

J. Birn
Textures

But what about this…

to this?

J. Birn
Textures

• How do we draw surfaces with complex detail?
Textures

• How do we draw surfaces with complex detail?

**Direct:**

• Tessellate in a complex manner and then associate the appropriate material properties to each vertex

Target Model

Complex Surface
Textures

• How do we draw surfaces with complex detail?

**Indirect:**
• Use a simple tessellation with an auxiliary texture image. Use the location of surface points to look up color values from the texture.
Textures

- Advantages:
  - The 3D model remains simple
  - It is easier to design/modify a texture image than it is to design/modify a surface in 3D.
Textures (2 dimensions)

Implementation:

• Associate a *texture coordinate* to each vertex \( v \):
  \[(s_v, t_v) \text{ with } (0 \leq s_v, t_v \leq 1)\]

• When rasterizing, *interpolate* to get the texture coordinate to at a pixel:
  \[(s_p, t_p)\]

• *Sample* the texture at \((s_p, t_p)\) to get the color at \( p \).
Example: Brick Wall
Example: Brick Wall

\[ (s_v, t_v) = (0,1) \quad (s_v, t_v) = (1,1) \]

\[ (s_v, t_v) = (0,0) \quad (s_v, t_v) = (1,0) \]
Textures (2 dimensions)

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

Given:
- A $d$-dimensional (multi-channel) texture image
- an assignment of coordinates $(s^1_v, ..., s^d_v)$ ranging over interval (0,1) to each vertex

At rasterization:
- Interpolate the texture coordinates to get the texture coordinate at each pixel, $(s^1_p, ..., s^d_p)$
- Sample the texture at $(s^1_p, ..., s^d_p)$ to get the value at $p$
Texture Mapping

• Scan conversion:
  ◦ Interpolate texture coordinates down/across scan lines

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

Linear interpolation of texture coordinates in screen space

Correct interpolation with perspective divide

Hill Figure 8.42
3D Rendering Pipeline (for direct illumination)

3D Primitives

Modeling Transformation

3D Modeling Coordinates

Camera Transformation

3D World Coordinates

Lighting

3D World Coordinates

Projection Transformation

3D Camera Coordinates

Clipping

2D Screen Coordinates

Viewport Transformation

2D Screen Coordinates

Scan Conversion

2D Image Coordinates

Image

Texture mapping
Overview

• Texture mapping methods
  ◦ Parameterization
  ◦ Filtering

• Texture mapping applications
  ◦ Modulation textures
  ◦ Illumination mapping
  ◦ Bump mapping
  ◦ Environment mapping
  ◦ Shadow maps
Map to a base domain

- Deform the geometry into a simple surface (with the same topology)
- Define a texture map over the simple surface

✓ Get a continuous map from the surface to the texture map
Map to a Base Domain

• Deform the geometry into a simple surface (with the same topology)

• Define a texture map over the simple surface

✓ Get a continuous map from the surface to the texture map

✗ Tricky, because mapped surface will have severe distortions

It is impossible to parameterize a complex shape to a simple domain so lengths are preserved
Map to a 2D Domain w/ Cuts

- Introduce cuts to give the surface a disk topology
- Map the cut surface to the 2D plane
- Assign texture coordinates in the plane

✓ Good cut placement can reduce distortion
✗ Need to ensure cross-seam continuity

[Piponi2000]
Texture Atlases

- Decompose the surface into multiple charts
- Map each chart to the 2D plane
- Assign texture coordinates in the plane

✓ Even less distortion in the mapping
✗ Even harder to ensure cross-seam continuity
✗ Need to pack the atlases into 2D

[Sander2001]
Overview

• Texture mapping methods
  ◦ Parameterization
  ◦ Filtering

• Texture mapping applications
  ◦ Modulation textures
  ◦ Illumination mapping
  ◦ Bump mapping
  ◦ Environment mapping
  ◦ Shadow maps
Texture Filtering

Sample texture to determine color at each pixel

Angel Figure 9.4
Texture Filtering

Sample texture to determine color at each pixel

- If the screen → texture transformation does not preserve areas, we need to compute the average of the pixels in texture space.

Average over many pixels
Texture Filtering

- Size of filter depends on the deformation

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

Average over many pixels
Mip Maps

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

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

Average over a few pixels

Texture hierarchy

Screen
Mip Maps

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

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

• Keep textures prefILTERED at multiple resolutions
  ◦ For each pixel, use the closest mip-map level(s)
  ◦ 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:

\[ \text{Sum}(\left[ a, b \right] \times \left[ c, d \right]) = \int_{a}^{b} \int_{c}^{d} f(x, y) \, dy \, dx \]
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:

\[
\int_a^b \int_c^d f(x,y) dy \, dx = \int_0^b \int_0^d f(x,y) dy \, dx - \int_0^b \int_0^c f(x,y) dy \, dx - \int_0^a \int_0^d f(x,y) dy \, dx + \int_0^a \int_0^c f(x,y) dy \, dx
\]
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:

\[
\int_a^b \int_c^d f(x, y) dy \, dx = \int_0^b \int_0^d f(x, y) dy \, dx - \int_0^b \int_0^c f(x, y) dy \, dx - \int_0^a \int_0^d f(x, y) dy \, dx + \int_0^a \int_0^c f(x, y) dy \, dx
\]
Key Idea:

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

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

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

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

\[
\int_a^b \int_c^d f(x, y) \, dy \, dx = \int_0^b \int_0^d f(x, y) \, dy \, dx - \int_0^b \int_0^c f(x, y) \, dy \, dx - \int_0^a \int_0^d f(x, y) \, dy \, dx + \int_0^a \int_0^c f(x, y) \, dy \, dx
\]
Summed-Area Tables (Pre-Process)

- 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

<table>
<thead>
<tr>
<th>Original image</th>
<th>Summed area table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 4 0</td>
<td></td>
</tr>
<tr>
<td>0 3 1 1</td>
<td></td>
</tr>
<tr>
<td>4 2 0 1</td>
<td></td>
</tr>
<tr>
<td>1 2 1 3</td>
<td></td>
</tr>
</tbody>
</table>
Summed-Area Tables (Pre-Process)

- 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
Summed-Area Tables (Pre-Process)

- 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

<table>
<thead>
<tr>
<th>Original image</th>
<th>Summed area table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  2  4  0</td>
<td>1  3</td>
</tr>
<tr>
<td>0  3  1  1</td>
<td></td>
</tr>
<tr>
<td>4  2  0  1</td>
<td></td>
</tr>
<tr>
<td>1  2  1  3</td>
<td></td>
</tr>
</tbody>
</table>

Original image: 1 2 4 0
Summed area table: 1 3
Summed-Area Tables (Pre-Process)

- 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

<table>
<thead>
<tr>
<th>Original image</th>
<th>Summed area table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  2  4  0</td>
<td>1 3  4</td>
</tr>
<tr>
<td>0  3  1  1</td>
<td></td>
</tr>
<tr>
<td>4  2  0  1</td>
<td></td>
</tr>
<tr>
<td>1  2  1  3</td>
<td></td>
</tr>
</tbody>
</table>
Summed-Area Tables (Pre-Process)

• 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

<table>
<thead>
<tr>
<th>Original image</th>
<th>Summed area table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 4 0</td>
<td>1 3 4 7</td>
</tr>
<tr>
<td>0 3 1 1</td>
<td></td>
</tr>
<tr>
<td>4 2 0 1</td>
<td></td>
</tr>
<tr>
<td>1 2 1 3</td>
<td></td>
</tr>
</tbody>
</table>
Summed-Area Tables (Pre-Process)

- 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
Summed-Area Tables (Pre-Process)

• 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

<table>
<thead>
<tr>
<th>Original image</th>
<th>Summed area table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 4 0</td>
<td>5 9</td>
</tr>
<tr>
<td>0 3 1 1</td>
<td></td>
</tr>
<tr>
<td>4 2 0 1</td>
<td>1 3 4 7</td>
</tr>
<tr>
<td>1 2 1 3</td>
<td></td>
</tr>
</tbody>
</table>

Original image

Summed area table
Summed-Area Tables (Pre-Process)

• 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

<table>
<thead>
<tr>
<th>Original image</th>
<th>Summed area table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 4 0</td>
<td>6 15 21 26</td>
</tr>
<tr>
<td>0 3 1 1</td>
<td>5 12 14 19</td>
</tr>
<tr>
<td>4 2 0 1</td>
<td>5 9 10 14</td>
</tr>
<tr>
<td>1 2 1 3</td>
<td>1 3 4 7</td>
</tr>
</tbody>
</table>
**Summed-Area Tables (Run-Time)**

- Given a pixel on the screen that maps to a rectangle in texture space, use the summed area table to compute the average:

  \[
  \text{Sum}([1,3] \times [2,3]) = S(3,3) - S(0,3) - S(3,1) + S(0,1) \\
  = 26 - 6 - 14 + 5 = 11
  \]

  \[
  \text{Average}([1,3] \times [2,3]) = \frac{\text{Sum}([1,3] \times [2,3])}{\text{Area}([1,3] \times [2,3])} = \frac{11}{6}
  \]

![Original image](image1.png)

![Summed-area table](image2.png)
Overview

• Texture mapping methods
  ◦ Parameterization
  ◦ Filtering

• Texture mapping applications
  ◦ Modulation textures
  ◦ Illumination mapping
  ◦ Bump mapping
  ◦ Environment mapping
  ◦ Shadow mapping
Modulation textures

Map texture values to scale factor

Modulation

\[ I = T(s, t) \left( I_E + K_A I_{AL} + \sum_i \left( K_D \langle \vec{N}, \vec{L}_i \rangle I_i + K_S \langle \vec{V}, \vec{R}_i \rangle^n I_i \right) \right) \]
Illumination Mapping

Map texture values to a material parameter

I = I_E + K_A I_{AL} + \sum_i \left[ T(s, t) N \cdot L_i I_i + K_S V \cdot R_i n I_i \right]
Illumination Mapping

Map texture values to a material parameter

Modulation

Diffuse

\[ I = I_E + K_A I_{AL} + \sum_i \left( T(s, t) \langle \hat{N}, \hat{L}_i \rangle I_i + K_S \langle \hat{V}, \hat{R}_i \rangle^n I_i \right) \]

Note that we need to evaluate the texture at each pixel but can still use the interpolated lighting values \( \langle \hat{N}, \hat{L}_i \rangle \)
Illumination Mapping

Map texture values to a material parameter

Modulation

Diffuse

Specular

\[ I = I_E + K_A I_{AL} + \sum_i \left( K_D \langle \vec{N}, \vec{L}_i \rangle I_i + T(s, t)(\vec{V}, \vec{R}_i)^n I_i \right) \]
Illumination Mapping

Map texture values to a material parameter

Again, we don’t need to re-compute most of the lighting calculation $\langle \vec{V}, \vec{R}_i \rangle^n$

\[
I = I_E + K_A I_{AL} + \sum_i \left( K_D \langle \vec{N}, \vec{L}_i \rangle I_i + T(s,t) \langle \vec{V}, \vec{R}_i \rangle^n I_i \right)
\]
Bump Mapping

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

\[ I = I_E + K_A I_{AL} + \sum_i \left( K_D \langle \vec{N}, \vec{L}_i \rangle I_i + K_S \langle \vec{V}, \vec{R}_i \rangle^n I_i \right) \]
Bump Mapping

Phong shading performs per-pixel lighting calculations with the interpolated normal
⇓
approximates a smoothly curved surface

Bump maps encode the normals in the texture
⇓
approximates a more complex undulating 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.
- Texture coordinates are computed dynamically through reflection.
Environment Mapping

- Generate a spherical/cubic map of the environment around the model.
- Texture coordinates are computed **dynamically** through reflection.

Set the texture coordinates based on the direction of the reflected view direction.
Environment Mapping

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

• Texture coordinates are computed **dynamically** through reflection

Set the texture coordinates based on the direction of the reflected view direction
Environment Mapping

- Generate a spherical/cubic map of the environment around the model.
- Texture coordinates are computed dynamically through reflection.

Set the texture coordinates based on the direction of the reflected view direction.
Environment Mapping

- Generate a spherical/cubic map of the environment around the model.
- Texture coordinates are computed dynamically through reflection

Set the texture coordinates based on the direction of the reflected view direction.

Changing the position of the camera changes the texture coordinates.
Environment Mapping

- Generate a spherical/cubic map of the environment around the model.
- Texture coordinates are computed dynamically through reflection.

Set the texture coordinates based on the direction of the reflected view direction.

Changing the position of the camera changes the texture coordinates.
Environment Mapping

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

- Texture coordinates are computed dynamically through reflection.

Set the texture coordinates based on the direction of the reflected view direction.

Changing the position of the camera changes the texture coordinates.
Environment Mapping

Texture coordinates are computed **dynamically** through reflection of the view direction through the surface normal
Shadow Mapping (Williams 1978)

Test if surface is visible to the light when computing the contribution the lighting equation.

Shadow Mapping (Williams 1978)

Test if surface is visible to the light when computing the contribution the lighting equation.

- Render the scene from the camera's perspective and read back the $z$-buffer/shadow map.

Shadow Mapping (Williams 1978)

Test if surface is visible to the light when computing the contribution the lighting equation.

• Render the scene from the camera's perspective and read back the $z$-buffer/shadow map.

• For each pixel in the camera view, compute it's coordinates relative to the light
  ◦ If it's further back than the value in the shadow map, it's in shadow
  ◦ Otherwise, it's illuminated

Shadow Mapping (Williams 1978)

Test if surface is visible to the light when computing the contribution the lighting equation.

- The projection used for rendering from the light-source depends on the type of light:
  - Directional → Orthographic
  - Point → Perspective

- Need to use multiple shadow maps if there are multiple lights in the scene

Shadow Mapping (Williams 1978)

Test if surface is visible to the light when computing the contribution the lighting equation.

- **Perspective Aliasing**: Stair-stepping due to limited shadow map resolution (particularly at grazing angles)

Shadow Mapping (Williams 1978)

Test if surface is visible to the light when computing the contribution the lighting equation.

- **Perspective Aliasing:**
  Stair-stepping due to limited shadow map resolution (particularly at grazing angles)

- **Shadow Acne:**
  Self-shadowing due to limited depth resolution