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…

J. Birn
Textures

• How do we draw surfaces with complex detail?

Target Model
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
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

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

\[(s_v, t_v) = (1,0) \quad (s_v, t_v) = (1,1)\]
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 $(s^1_v, \ldots, s^d_v)$ ranging over interval (0,1) to each vertex

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

Linear interpolation of texture coordinates in screen space

Correct interpolation with perspective divide

Hill Figure 8.42
3D Rendering Pipeline

- **3D Primitives**
- **Modeling Transformation**
- **Viewing Transformation**
- **Lighting**
- **Projection Transformation**
- **Clipping**
- **Viewport Transformation**
- **Scan Conversion**

- **3D Modeling Coordinates**
- **3D World Coordinates**
- **3D Camera Coordinates**
- **2D Screen Coordinates**
- **2D Image Coordinates**

- Perform the texture interpolation and look-up while rasterizing the pixels.

```
(s1, t1)
A

\(\alpha\)

\(\beta\)

\(\gamma\)

\((s,t)\)

B

(\(s_3, t_3\))
```

```
(\(s_2, t_2\))
```

**texture mapping**
Overview

- Texture mapping methods
  - Parameterization
  - Sampling

- 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 some distortion
✗ Need to ensure cross-seam continuity
✗ Still have to contend with distortion

[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
  ◦ Sampling

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

Given pixel on a screen:
1. Determine the corresponding surface patch
2. Determine the corresponding texture patch
Texture Filtering

Given pixel on a screen:

1. Determine the corresponding surface patch
2. Determine the corresponding texture patch
3. Average texel values over the texture patch
Texture Filtering

× Size of texture patch depends on the deformation
  ◦ Computation is linear in the size of the pixel footprint

• Can pre-filter images for better performance
  ◦ MIP (Multum In Parvo) maps
  ◦ Summed area tables

Average over many pixels

Texture

Screen
MIP Maps

- In a *pre-processing* step, compute a hierarchy of successively down-sampled texture images
MIP Maps

- In a pre-processing step, compute a hierarchy of successively down-sampled texture images.
- At run-time, sample the closest MIP map level(s):
  - Easy for hardware
  - Computation is constant in the size of the pixel footprint.
MIP Maps

• In a pre-processing step, compute a hierarchy of successively down-sampled texture images

✓ Storage is only 4/3 the size of the input image
MIP Maps

• In a pre-processing step, compute a hierarchy of successively down-sampled texture images
  ✓ Storage is only 4/3 the size of the input image

• At run-time, sample the closest MIP map level(s)
  ✗ This type of filtering is isotropic:
    » Assumes there is the same amount of compression along the vertical and horizontal directions

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:

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

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

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

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

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

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

- 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 summed-area table texel is the sum of all input texels below and to the left

\[
\begin{array}{cccc}
1 & 2 & 4 & 0 \\
0 & 3 & 1 & 1 \\
4 & 2 & 0 & 1 \\
1 & 2 & 1 & 3 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{Input image} & \text{Summed area table} \\
\end{array}
\]
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 summed-area table texel is the sum of all input texels below and to the left

<table>
<thead>
<tr>
<th>Input 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>1</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 summed-area table texel is the sum of all input texels below and to the left
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 summed-area table texel is the sum of all input texels below and to the left

<table>
<thead>
<tr>
<th>Input 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 summed-area table texel is the sum of all input texels below and to the left

<table>
<thead>
<tr>
<th>Input 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>
<tr>
<td></td>
<td>1 3 4 7</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 summed-area table texel is the sum of all input texels below and to the left
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 summed-area table texel is the sum of all input texels below and to the left

<table>
<thead>
<tr>
<th>Input 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>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 9</td>
</tr>
<tr>
<td></td>
<td>1 3 4 7</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 summed-area table texel is the sum of all input texels below and to the left

<table>
<thead>
<tr>
<th>Input 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)

Example:

Compute the average in the rectangle $[1,3] \times [2,3]$. 
⇒ Compute the sum and divide by the area

Input image

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Summed-area table

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

Example:

Compute the average in the rectangle $[1,3] \times [2,3]$. 
⇒ Compute the sum and divide by the area

$\text{Sum}([1,3] \times [2,3]) = S(3,3)$
Summed-Area Tables (Run-Time)

Example:

Compute the average in the rectangle $[1,3] \times [2,3]$. 
⇒ Compute the sum and divide by the area 
$\text{Sum}([1,3] \times [2,3]) = S(3,3) - S(0,3)$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Input image

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>15</th>
<th>21</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>14</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>10</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Summed-area table
### Summed-Area Tables (Run-Time)

**Example:**

Compute the average in the rectangle $[1,3] \times [2,3]$. ⇒ Compute the sum and divide by the area

$$\text{Sum}( [1,3] \times [2,3]) = S(3,3) - S(0,3) - S(3,1)$$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

**Input image**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>15</td>
<td>21</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>14</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>10</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

**Summed-area table**
Summed-Area Tables (Run-Time)

Example:

Compute the average in the rectangle $[1,3] \times [2,3]$. 

$\Rightarrow$ Compute the sum and divide by the area 

$$\text{Sum}([1,3] \times [2,3]) = S(3,3) - S(0,3) - S(3,1) + S(0,1)$$
Summed-Area Tables (Run-Time)

Example:

Compute the average in the rectangle $[1,3] \times [2,3]$.

$\Rightarrow$ Compute the sum and divide by the area

$$\text{Sum}([1,3] \times [2,3]) = S(3,3) - S(0,3) - S(3,1) + S(0,1)$$

$$= 26 - 6 - 14 + 5 = 11$$

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Input image

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>15</th>
<th>21</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>14</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>10</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Summed-area table
Summed-Area Tables (Run-Time)

- Precompute the values of the integral

✓ Constant time averaging, regardless of rectangle size

✗ If the input image has values in the range \([0, 255]\) (i.e. one byte per channel), the summed area table can have values in the range \([0, 255 \cdot \text{width} \cdot \text{height}]\)

<table>
<thead>
<tr>
<th></th>
<th>Input image</th>
<th>Summed-area table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>26</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>14</td>
</tr>
</tbody>
</table>
Overview

• Texture mapping methods
  ○ Parameterization
  ○ Sampling

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

Map texture values to scale factor

Modulation

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

Map texture values to a material parameter

Modulation

Diffuse

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

Map texture values to a material parameter

Modulation

Diffuse

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

This requires the graphics card to separately store the diffuse component of the lighting at each vertex

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

Map texture values to a material parameter

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

Map texture values to a material parameter

Modulation

Diffuse

Specular

Again, we can use the interpolated lighting values

\[
\langle \hat{V}, \hat{R}_i \rangle^n
\]

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

\[ \downarrow \]

approximates a smoothly curved surface

Bump maps encode the normals in the texture

\[ \downarrow \]

approximates a more complex undulating surface

P. Rheingans
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.

At the same triangle, 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.

At the same triangle, 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.

At the same triangle, 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)

Q: Is the point that is seen by the camera visible to the light?

A: The point is visible if:

- It’s as close to the light as the closest primitive in the scene.
  ⇒ In the light’s coordinate system, the point’s $z$-coordinate equals the $z$-coordinate of the primitive seen by the light.
  ⇒ Rendering the scene from the light’s perspective, the point’s $z$-coordinate equals the value stored in the $z$-buffer.
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 its 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

Images courtesy of