Texture filtering

UCSD CSE 168
Rendering
Tzu-Mao Li
Last time: texture mapping

- assign a “UV” 2D vector to each point on the surface
Reconstructing a location on a texture

\[ f(u, v) = \sum_i w_i F(u_i, v_i) \]

interpolating from nearby discrete values
For now, let’s assume we use bilinear interpolation

\[ f(u, v) = (1 - \Delta_u)(1 - \Delta_v)F_{00} + \Delta_u(1 - \Delta_v)F_{10} + (1 - \Delta_u)\Delta_vF_{01} + \Delta_u\Delta_vF_{11} \]
Pixel filter support can cover a large region in the texture.
Fail to account for all texels in the region can lead to noise/aliasing.
Goal: (weighted) average all texture values inside the region
The filtering region is determined by the mapping between image space and texture space.

**quiz:** what is the mapping $T$?

\[ u, v = T(x, y) \]
We can approximate the local mapping using first-order Taylor expansion

$$u, v \approx T(x_0, y_0) + \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

**quiz:** how do we determine $\Delta x$ & $\Delta y$?
We can approximate the local mapping using first-order Taylor expansion:

\[ u, v \approx T(x_0, y_0) + \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \]

approximate local regions using an ellipse

“Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter”, Greene and Heckbert 1986
Goal: (weighted) average all texture values inside the region

\[ u, v \approx T(x_0, y_0) + \left[ \begin{array}{cc} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{array} \right] \left[ \begin{array}{c} \Delta x \\ \Delta y \end{array} \right] \]

- how do we obtain the derivatives?
- how do we (quickly) compute the weighted average inside the ellipse?
Goal: (weighted) average all texture values inside the region

\[ u, v \approx T(x_0, y_0) + \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \]

- how do we obtain the derivatives?
- how do we (quickly) compute the weighted average inside the ellipse?
The mapping $T$ is the ray casting function

- so we can apply chain rule and differentiate ray casting

$$u, v \approx T(x_0, y_0) + \left[ \begin{array}{cc} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{array} \right] \left[ \begin{array}{c} \Delta x \\ \Delta y \end{array} \right]$$
The mapping $T$ is the ray casting function

- so we can apply chain rule and differentiate ray casting

$$x, y \xrightarrow{f_1} D \xrightarrow{f_2} b_1, b_2 \xrightarrow{f_3} u, v$$

$$u, v = f_3 \left( f_2 \left( f_1 \left( x, y \right) \right) \right)$$
The mapping $T$ is the ray casting function

- so we can apply chain rule and differentiate ray casting

\[ x, y \xrightarrow{f_1} D \xrightarrow{f_2} b_1, b_2 \xrightarrow{f_3} u, v \]

\[ u, v = f_3 \left( f_2 \left( f_1 (x, y) \right) \right) \]

\[ \frac{\partial(u, v)}{\partial(x, y)} = \frac{df_3}{df_2} \frac{df_2}{df_1} \frac{df_1}{dx \, dy} \]
Tracing Ray Differentials

\[ \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \]

\[ u, v = f_3 \left( f_2 \left( f_1 \left( x, y \right) \right) \right) \]

\[ \frac{duv}{dxy} = \frac{df_3}{df_2} \frac{df_2}{df_1} \frac{df_1}{dxy} \]
Tracing Ray Differentials

D(x, y) = \frac{\text{view} + x \times \text{right} + y \times \text{up}}{\| \text{view} + x \times \text{right} + y \times \text{up} \|}

what is \frac{\partial D}{\partial x}?
Tracing Ray Differentials

\[ D(x, y) = \frac{\text{view} + x \cdot \text{right} + y \cdot \text{up}}{\| \text{view} + x \cdot \text{right} + y \cdot \text{up} \|} \]

\[ \frac{\partial D}{\partial x} = \frac{\| d \| \cdot \text{right} - (d \cdot \text{right}) d}{\| d \|^3} \]

\[ d = \text{view} + x \cdot \text{right} + y \cdot \text{up} \]
Tracing Ray Differentials

\[ u, v = f_3 \left( f_2 \left( f_1 \left( x, y \right) \right) \right) \]

\[ \frac{\partial u}{\partial x} \frac{\partial u}{\partial y} \frac{\partial v}{\partial x} \frac{\partial v}{\partial y} \]

\[
\begin{vmatrix}
\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\
\frac{\partial v}{\partial x} & \frac{\partial v}{\partial y}
\end{vmatrix}
\]

\[ \frac{\partial uv}{\partial xy} = \frac{df_3}{df_2} \frac{df_2}{df_1} \frac{df_1}{dxy} \]
Tracing Ray Differentials

Goal: find

\[
\begin{bmatrix}
\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\
\frac{\partial v}{\partial x} & \frac{\partial v}{\partial y}
\end{bmatrix}
\]

Quiz: how do we compute \(b_1, b_2, t\)?
Ray-triangle intersection

\[ P = O + t \cdot D \]

\[ P = (1 - b_1 - b_2)P_0 + b_1 P_1 + b_2 P_2 \]

Goal: find \[
\begin{bmatrix}
\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\
\frac{\partial v}{\partial x} & \frac{\partial v}{\partial y}
\end{bmatrix}
\]

https://en.wikipedia.org/wiki/Barycentric_coordinate_system
Ray-triangle intersection

\[ O + tD = (1 - b_1 - b_2)P_0 + b_1 P_1 + b_2 P_2 \]

\[
\begin{bmatrix} P_1 - P_0, P_2 - P_0, -D \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ t \end{bmatrix} = O - P_0
\]
Ray-triangle intersection

\[ O + tD = (1 - b_1 - b_2)P_0 + b_1P_1 + b_2P_2 \]

\[
\begin{align*}
    b_1 &= \frac{\left( D \times (P_2 - P_0) \right) \cdot (O - P_0)}{\left( D \times (P_2 - P_0) \right) \cdot (P_1 - P_0)} \\
    b_2 &= \frac{\left( (O - P_0) \times (P_1 - P_0) \right) \cdot D}{\left( D \times (P_2 - P_0) \right) \cdot (P_1 - P_0)}
\end{align*}
\]
Ray-triangle intersection

\[ O + tD = (1 - b_1 - b_2)P_0 + b_1 P_1 + b_2 P_2 \]

\[ b_1 = \frac{ \left( D \times (P_2 - P_0) \right) \cdot (O - P_0)}{\left( D \times (P_2 - P_0) \right) \cdot (P_1 - P_0)} \]

\[ b_2 = \frac{ \left( (O - P_0) \times (P_1 - P_0) \right) \cdot D}{\left( D \times (P_2 - P_0) \right) \cdot (P_1 - P_0)} \]

**quiz:** given \( \frac{\partial D}{\partial x} \), what are \( \frac{\partial b_1}{\partial x} \) and \( \frac{\partial b_2}{\partial x} \)?
Tracing Ray Differentials  

Goal: find \[ \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \]

\[ x, y \xrightarrow{D} f_1 \xrightarrow{f_2} b_1, b_2 \xrightarrow{f_3} u, v \]

Ray direction  
Barycentric coordinates

\[ u, v = f_3 \left( f_2 \left( f_1 \left( x, y \right) \right) \right) \]

\[ \frac{d[u,v]}{d[x,y]} = \begin{bmatrix} df_3 & df_2 & df_1 \\ df_2 & df_1 & dxy \end{bmatrix} \]
From barycentric coordinates to UV

\[ u = (1 - b_1 - b_2)u_0 + b_1 u_1 + b_2 u_2 \]
\[ v = (1 - b_1 - b_2)v_0 + b_1 v_1 + b_2 v_2 \]

**goal:** find \( \frac{\partial u}{\partial x} \) and \( \frac{\partial v}{\partial y} \)

**quiz:** given \( \frac{\partial b_i}{\partial x} \), what are \( \frac{\partial u}{\partial x} \) and \( \frac{\partial v}{\partial y} \)?
Recap

- apply chain rule and differentiate ray casting

\[ u, v = f_3 \left( f_2 \left( f_1(x, y) \right) \right) \]

\[ \frac{d(u, v)}{d(x, y)} = \frac{df_3}{df_2} \frac{df_2}{df_1} \frac{df_1}{d(x, y)} \]
Further reading


10.1.1 Finding the Texture Sampling Rate

Consider an arbitrary texture function that is a function of position, $T(p)$, defined on a surface in the scene. If we ignore the complications introduced by visibility issues—the possibility that another object may occlude the surface at nearby image samples or that the surface may have a limited extent on the image plane—this texture function can also be expressed as a function over points $(x, y)$ on the image plane, $T(f(x, y))$, where $f(x, y)$ is the function that maps image points to points on the surface. Thus, $T(f(x, y))$ gives the value of the texture function as seen at image position $(x, y)$.

As a simple example of this idea, consider a 2D texture function $T(s, t)$ applied to a quadrilateral that is perpendicular to the $z$ axis and has corners at the world space points $(0, 0, 0), (1, 0, 0), (1, 1, 0)$, and $(0, 1, 0)$. If an orthographic camera is placed looking down the $z$ axis such that the quadrilateral precisely fills the image plane and if points $p$ on the quadrilateral are mapped to 2D $(s, t)$ texture coordinates by

$$
\begin{align*}
    s &= p_x \\
    t &= p_y,
\end{align*}
$$

then the relationship between $(s, t)$ and screen $(x, y)$ pixels is straightforward:

$$
\begin{align*}
    s &= \frac{x}{x_i} \\
    t &= \frac{y}{y_i},
\end{align*}
$$

where the overall image resolution is $(x_i, y_i)$ (Figure 10.2). Thus, given a sample spacing of one pixel in the image plane, the sample spacing in $(s, t)$ texture parameter space is $(1/x_i, 1/y_i)$, and the texture function must remove any detail at a higher frequency than can be represented at that sampling rate.
You don’t need to do this!

Automatic differentiation can turn code into derivatives

- operator overloading
- or a customized compiler

```cpp
ADFloat f(const ADFloat x[2]) {
  ADFloat y = sin(x[0]);
  ADFloat z = cos(x[1]);
  return y * z;
}
```
Goal: (weighted) average all texture values inside the region

\[
u, v \approx T(x_0, y_0) + \begin{bmatrix}
\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\
\frac{\partial v}{\partial x} & \frac{\partial v}{\partial y}
\end{bmatrix} \begin{bmatrix}
\Delta x \\
\Delta y
\end{bmatrix}
\]

- how do we obtain the derivatives?
- how do we (quickly) compute the weighted average inside the ellipse?
Downsample the texture for fast average

- usually called “mipmapping”

**quiz:** what is the extra storage cost?

<table>
<thead>
<tr>
<th>Size</th>
<th>512x512</th>
<th>256x256</th>
<th>128x128</th>
<th>64x64</th>
<th>32x32</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0,1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1,0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1,1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Elliptical weighted averaging algorithm (the FELINE/Texram variant)

Elliptical weighted averaging algorithm

1. approximate the ellipse with circles

Elliptical weighted averaging algorithm

1. approximate the ellipse with circles

2. find two appropriate mipmap levels s.t. each circle maps to ~1 texel

Elliptical weighted averaging algorithm

1. approximate the ellipse with circles

2. find two appropriate mipmap levels s.t. each circle maps to ~1 texel

3. linearly interpolate between pixels and mipmap levels

Elliptical weighted averaging algorithm

1. approximate the ellipse with circles

2. find two appropriate mipmap levels s.t. each circle maps to ~1 texel

3. linearly interpolate between pixels and mipmap levels

Goal: approximate an ellipse with circles

an ellipse has a major axis $A$ and a minor axis $B$

define “anisotropy” as $\frac{\|A\|}{\|B\|}$

quiz: if anisotropy is very low, how many circles do we need?
Goal: approximate an ellipse with circles

an ellipse has a major axis $A$ and a minor axis $B$

define “anisotropy” as $\frac{\| A \|}{\| B \|}$

choose number of circles $= \text{round} \left( 2 \times \text{anisotropy} - 1 \right)$

(cap the number of circles to a maximum user-defined value)
Goal: approximate an ellipse with circles

choose radius = minor axis radius
place the circles evenly across the major axis
Elliptical weighted averaging algorithm
(the FELINE/Texram variant)

1. approximate the ellipse with circles

2. find two appropriate mipmap levels s.t. each circle maps to ~1 texel

3. linearly interpolate between pixels and mipmap levels

Goal: given a circle, find mipmap level

radius=5 texels, which level should we choose?
Goal: given a circle, find mipmap level

radius = 5 texels, which level should we choose?

choose level = floor(log₂ radius) and ceil(log₂ radius)
Elliptical weighted averaging algorithm

1. approximate the ellipse with circles

2. find two appropriate mipmap levels s.t.
   each circle maps to ~1 texel

3. linearly interpolate between pixels and
   mipmap levels

Interpolate between mipmap levels

choose level = floor(log₂ radius) and ceil(log₂ radius),
interpolate using log₂ radius

"trilinear interpolation"
Modern video games/renderers use EWA filtering!

Comparison: point sampling

image from Steve Marschner
Comparison: 512x anti-aliased, Gaussian pixel filter

image from Steve Marschner
Comparison: 1 pixel sample with mipmapping

Also work for specular reflection/refraction
Next: normals and displacements