Image Warping

Source image → Warp → Destination image
Image Mapping

- Define transformation
  - Describe the destination \((x,y)\) for every location \((u,v)\) in the source (or vice-versa, if invertible)
Image Mapping - Examples

- Scale by factor:
  - \( x = \text{factor} \times u \)
  - \( y = \text{factor} \times v \)

- Rotate by \( \Theta \) degrees:
  - \( x = u \cos \Theta - v \sin \Theta \)
  - \( y = u \sin \Theta + v \cos \Theta \)

- Shear in X by factor:
  - \( x = u + \text{factor} \times v \)
  - \( y = v \)

- Shear in Y by factor:
  - \( x = u \)
  - \( y = v + \text{factor} \times u \)

- Any function of \( u \) and \( v \):
  - \( x = f_x(u,v) \)
  - \( y = f_y(u,v) \)

Examples:
- Scale 0.8
- Rotate 30 degrees
- Shear X 1.3
- Shear Y 1.3
- Fish-eye
- "Swirl"
- "Rain"
\[
\begin{bmatrix}
x^* \\
y^*
\end{bmatrix} =
\begin{bmatrix}
1 & -\tan(\theta/2) \\
0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
1 & 0 \\
\sin \theta & 1 \\
\end{bmatrix}
\begin{bmatrix}
1 & -\tan(\theta/2) \\
0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
\]

\[
\begin{bmatrix}
\cos \theta & \sin \theta \\
-\sin \theta & \cos \theta
\end{bmatrix}
\]
Forward Mapping
Forward Mapping

```c
for (int u = 0; u < umax; u++) {
    for (int v = 0; v < vmax; v++) {
        float x = f_x(u, v);
        float y = f_y(u, v);
        dst(x, y) = src(u, v);
    }
}
```
Forward Mapping - Disadvantages

Many source pixels can map to same destination pixel.

Some destination pixels may not be covered.
Example – Forward Mapping

Original

Rotated

Zoom In
Backward Mapping

```c
for (int x = 0; x < xmax; x++) {
    for (int y = 0; y < ymax; y++) {
        float u = f_x^{-1}(x, y);
        float v = f_y^{-1}(x, y);
        dst(x, y) = src(u, v);
    }
}
```

The Problem:
(u,v) are not integers!
Nearest Neighbor

- Take value at closest pixel:
  - int iu = trunc(u+0.5);
  - int iv = trunc(v+0.5);
  - dst(x,y) = src(iu,iv);

This method is simple, but it causes aliasing.
Example - Nearest Neighbor

Original

Rotated

Zoom In
Bi-linear

- Bi-linear interpolates four closest pixels.
- The weight for each pixel is proportional to its distance from the sampling point \((x,y)\)
Bi-linear Interpolation
Bi-linear Interpolation

Model $f(x, y)$ as a bilinear surface
Bi-linear Interpolation

- Model $f(x, y)$ as a bilinear surface
- Interpolate $f\left(\frac{1}{2}, 0\right)$ using $f(0, 0)$ and $f(1, 0)$
- Interpolate $f\left(\frac{1}{2}, 1\right)$ using $f(0, 1)$ and $f(1, 1)$
Bi-linear Interpolation

Model $f(x, y)$ as a bilinear surface.

Interpolate $f(\frac{1}{2}, 0)$ using $f(0, 0)$ and $f(1, 0)$.
Interpolate $f(\frac{1}{2}, 1)$ using $f(0, 1)$ and $f(1, 1)$.
Interpolate $f(\frac{1}{2}, \frac{1}{2})$ using $f(\frac{1}{2}, 0)$ and $f(\frac{1}{2}, 1)$. 

Example Bi-linear

Original

Rotated

Zoom In
Bi-cubic

- Bicubic interpolates 16 closest neighbors (4x4 neighborhood)
  - The result is much more smooth
Bi-cubic Interpolation
Bi-cubic Interpolation
Bi-cubic Interpolation

- Interpolate
  - $f\left(\frac{1}{2}, 0\right)$ using $f(0, 0)$, $f(1, 0)$, $\partial_x f(0, 0)$ and $\partial_x f(1, 0)$
  - $f\left(\frac{1}{2}, 1\right)$ using $f(0, 1)$, $f(1, 1)$, $\partial_x f(0, 1)$ and $\partial_x f(1, 1)$
  - $\partial_y f\left(\frac{1}{2}, 0\right)$ using $\partial_y f(0, 0)$, $\partial_y f(1, 0)$, $\partial_{xy} f(0, 0)$ and $\partial_{xy} f(1, 0)$
  - $\partial_y f\left(\frac{1}{2}, 1\right)$ using $\partial_y f(0, 1)$, $\partial_y f(1, 1)$, $\partial_{xy} f(0, 1)$ and $\partial_{xy} f(1, 1)$
  - Interpolate $f\left(\frac{1}{2}, \frac{1}{2}\right)$ using $f\left(\frac{1}{2}, 0\right)$, $f\left(\frac{1}{2}, 1\right)$, $\partial_y f\left(\frac{1}{2}, 0\right)$ and $\partial_y f\left(\frac{1}{2}, 1\right)$
Example Bi-cubic

Original

Rotated

Zoom In
Nearest Neighbor
Bi-Linear Interpolation
Bi-Cubic Interpolation
Comparison

Nearest Neighbor

Bi-linear

Bi-cubic
Nearest neighbor sampling
Filtered Texture:
Texture Aliasing

- A single screen space pixel might correspond to many texels (texture elements):
Texture Mapping
Texture Pre-Filtering

• **Problem**: filtering the texture during rendering is too slow for interactive performance.

• **Solution**: pre-filter the texture in advance

  – Summed area tables - gives the average value of each axis-aligned rectangle in texture space

  – Mip-maps (tri-linear interpolation) - supported by most of today’s texture mapping hardware
MIP-Maps

• Precompute a set of prefiltered textures (essentially an image pyramid).

• Based on the area of the pre-image of the pixel:
  – Select two “best” resolution levels
  – Use bilinear interpolation inside each level
  – Linearly interpolate the results

• Referred to as trilinear interpolation
MIP Maps
MIP Mapping

• Lance Williams, 1983

• Create a resolution pyramid of textures
  – Repeatedly subsample texture at half resolution
  – Until single pixel
  – Need extra storage space

• Accessing
  – Use texture resolution closest to screen resolution
  – Or interpolate between two closest resolutions
Texture Aliasing

• Image mapped onto polygon
• Occur when screen resolution differs from texture resolution

• **Magnification aliasing**
  – Screen resolution finer than texture resolution
  – Multiple pixels per texel

• **Minification aliasing**
  – Screen resolution coarser than texture resolution
  – Multiple texels per pixel
Minification Filtering

• Multiple texels per pixel
• Potential for aliasing since texture signal bandwidth greater than image
• Box filtering requires averaging of texels
• Precomputation
  – MIP Mapping
  – Summed Area Tables
Summed Area Table

• Frank Crow, 1984
• Replaces texture map with summed-area texture map
  – \( S(x,y) = \text{sum of texels} \leq x,y \)
  – Need double range (e.g. 16 bit)
• Creation
  – Incremental sweep using previous computations
  – \( S(x,y) = T(x,y) + S(x-1,y) + S(x,y-1) - S(x-1,y-1) \)
• Accessing
  – \( \Sigma T([x_1,x_2],[y_1,y_2]) = S(x_2,y_2) - S(x_1,y_2) - S(x_2,y_1) + S(x_1,y_1) \)
  – Ave \( T([x_1,x_2],[y_1,y_2]) / ((x_2 - x_1)(y_2 - y_1)) \)
Summed Area Tables

• A 2D table the size of the texture. At each entry \((i,j)\), store the sum of all texels in the rectangle defined by \((0,0)\) and \((i,j)\).

• Given any axis aligned rectangle, the sum of all texels is easily obtained from the summed area table:

\[
\text{area} = A - B - C + D
\]
Quality considerations

- Pixel area maps to "weird" (warped) shape in texture space
Mip-maps

• Find level of the mip-map where the area of each mip-map pixel is closest to the area of the mapped pixel.
Summed Area Table (SAT)

- Determining the rectangle:
  - Find bounding box and calculate its aspect ratio
Summed Area Table (SAT)

- Determine the rectangle with the same aspect ratio as the bounding box and the same area as the pixel mapping.
Elliptical Weighted Average (EWA) Filter

- Treat each pixel as circular, rather than square.
- Mapping of a circle is elliptical in texel space.
Texture Domain
Elliptical Weighted Average

Bounding box enclosing the contours $Q = F$