Warping & Sampling

Outline
• What is image warping?
  • Magnification examples
• Warping approaches:
  • Forward warping
  • Backward warping
• Resampling methods:
  • Point sampling
  • Bilinear interpolation
  • Gaussian filtering

What is Image Warping
• Distort the original image by moving pixels to new location
  • Define the mapping relation
  • Resample the original image

Circle Preserving Magnification
• Warping function:
  • \( r = (u^2 + v^2)^{1/2} \)
  • \( s = u / r^{1/2} \)
  • \( t = v / r^{1/2} \)
• Inverse function:
  • \( r = (s^2 + t^2)^{1/2} \)
  • \( u = r * s \)
  • \( v = r * t \)

Line Preserving Magnification
• Warping function:
  • \( s = \sin(u * n / 2) \)
  • \( t = \sin(v * n / 2) \)
• Inverse function:
  • \( u = 2 * \arcsin(s) / n \)
  • \( v = 2 * \arcsin(t) / n \)

Forward Warping
• \( (s,t) = F(u,v) \)
  • Describe the destination \( (s,t) \) for every pixel \( (u,v) \)
    in the source
  • Map from known to unknown
• If \( F \) is invertible:
  • \( G(x,y) = F^{-1}(x,y) \)
  • \( G \) is the corresponding backward warping function
Limitations of Forward Warping

- Iterate over pixels in source image
- More than one source pixels may map to same destination pixel
- Some destination pixels may not be covered

Backward Warping

\((u,v) = G(s,t)\)

- Describe the source \((u,v)\) for every pixel \((s,t)\) in the destination
- Map from unknown to known
- If \(G\) is invertible:
  \(F(x,y) = G^{-1}(x,y)\)
  \(F\) is the corresponding forward warping function

Implementation Comparison

// Forward warping:
for (int v=0; v<height; v++)
  for (int u=0; u<width; u++) {
    float s = Fx(u,v);
    float t = Fy(u,v);
    destination( round(s), round(t) ) = source(u,v);
  }

// Backward warping:
for (int t=0; t<height; t++)
  for (int s=0; s<width; s++) {
    float u = Gx(s,t);
    float v = Gy(s,t);
    destination(s,t) = sampleSource(u,v);
  }

Resampling Source Image

- How to compute the color at in-between pixel locations?
  - Point sampling
  - Linear interpolation
  - Gaussian filtering

Point Sampling

- Simply use the closest pixel
  \(\text{sampleSource}(u,v)\) {
    return source( round(u), round(v) );
  }

- Problems:
  - Subject to blocky and aliasing artifacts

Blocky Artifact

- Multiple pixels in the destination image have the same color
  - Due to locally increase image resolution (oversampling)
  - Can be reduced by bilinear interpolation
Bilinear Interpolation

- Interpolate among nearby 4 pixels
  - First interpolate along horizontal direction to obtain colors at positions A & B
  - Then interpolate along vertical direction using A & B
- Result is weighted sum of the 4 pixels' colors
  - Closer pixel has higher weight

Pseudocode for Bilinear Interpolation

```java
sampleSourceLinear(float u, float v) {
    int x = floor(u);
    int y = floor(v);
    color = source(x, y) * (x+1-u) * (y+1-v);
    color += source(x+1, y) * (u-x) * (y+1-v);
    color += source(x, y+1) * (x+1-u) * (v-y);
    color += source(x+1, y+1) * (u-x) * (v-y);
    return color;
}
```

Aliasing Artifact

- Details in the source image are lost or become random noise
  - Due to locally decrease resolution (undersampling)
  - Can be reduced by Gaussian filtering

2D Gaussian Function

- Definition:

- Parameters:
  - \((u,v)\) is the center, where the function has the maximum value
  - \(\sigma\) is the standard deviation, which controls the width of the “bell”

Gaussian Filtering

- Compute weighted sum of neighboring pixels
  - Weights are normalized values of Gaussian function
  - Number of neighbors involved depends on \(\sigma\)
    - The larger the \(\sigma\) is, the more blurry the result is

Pseudocode for Gaussian Filtering

```java
sampleSourceGaussian(float u, float v) {
    color = 0; sum = 0;
    for (int y=floor(v)-r, y<=ceil(v)+r, y++)
        for (int x=floor(u)-r, x<=ceil(u)+r, x++) {
            float w = Gaussian(x, y, u, v, \sigma);
            color += source(x, y) * w;
            sum += w;
        }
    color /= sum;
    return color;
}
```