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:
  \[ \begin{align*}
  r &= u^2 + v^2 \\
  s &= u / r \\
  t &= v / r
  \end{align*} \]
- Inverse function:
  \[ \begin{align*}
  r &= s^2 + t^2 \\
  u &= rs \\
  v &= rt
  \end{align*} \]

Line Preserving Magnification
- Warping function:
  \[ \begin{align*}
  s &= \sin \left( \frac{u}{2} \right) \\
  t &= \sin \left( \frac{v}{2} \right)
  \end{align*} \]
- Inverse function:
  \[ \begin{align*}
  u &= 2 \arcsin(s) \\
  v &= 2 \arcsin(t)
  \end{align*} \]

Forward Warping
- Directly apply the warping function
  \[ (s, t) = F(u, v) \]
- Describe the destination \((s, t)\) for every pixel \((u, v)\) in the source
- Map from known to unknown

```java
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);
  }
```
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

- Works when $F(u, v)$ is invertible
- $G(x, y) = F^{-1}(x, y)$ is the corresponding backward warping function
- $(u, v) = G(x, y)$
- Describe the source $(u, v)$ for every pixel $(x, y)$ in the destination
- Map from unknown to known

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
  - `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**

```c
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:**
  \[ \mathcal{G}(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}} \]

- **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**

```c
sampleSourceGaussian(float u, float v) {
    color = 0; sum = 0;
    for (int y=ceil(v)-r, y<=ceil(v)+r, y++)
        for (int x=ceil(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;
}
```