Warping & Sampling

Outline
- Mapping function:
  - Forward warping
  - Backward warping
- Re-sampling:
  - 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.

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.

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.

Example: Magnify Function I
- Forward warping function:
  - \(r = (u^2 + v^2)^{1/2}\); \(s = u / r^2\)
  - \(t = v / r^2\)
  - Feature:
    - Preserve circles
- Inverse function:
  - \(r = (s^2 + t^2)^{1/2}\)
  - \(u = r * s\)
  - \(v = r * t\)
Example: Magnify Function II

- **Forward warping function:**
  - \[ s = \sin(u \cdot \pi / 2); \]
  - \[ t = \sin(v \cdot \pi / 2); \]
- **Feature:**
  - Preserve straight lines

---

Implementation Comparison

- // Forward warping:
  - for (int v=0; v<height; v++)
    - for (int u=0; u<width; u++)
      - \[ s = F_x(u,v); \]
      - \[ t = F_y(u,v); \]
      - \[ destination(round(s), round(t)) = source(u, v); \]

- // Backward warping:
  - for (int t=0; t<height; t++)
    - for (int s=0; s<width; s++)
      - \[ u = 2 \times \text{asin}(s) / \pi; \]
      - \[ v = 2 \times \text{asin}(t) / \pi; \]

---

Difference?

- **Forward warping**
  - Iterate over source image.
  - More than one source pixels may map to same destination pixel.
  - Some destination pixels may not be covered.

- **Backward warping**
  - Iterate over destination image.
  - All destination pixels are covered.
  - How to resample source image?

---

Point Sampling

- Simply use the closest pixel.
  - \[ \text{source}(u, v) = \begin{cases} \text{source}(\text{round}(u), \text{round}(v)) & \text{for } (u,v) \in \Omega \setminus \partial \Omega \setminus \emptyset \\ \text{linear interpolate} \end{cases} \]
- **Problems:**
  - Subject to blocky and aliasing artifacts.

---

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
int x = floor(u);
int y = floor(v);
float color = source(x,y) * (x+1-u) * (y+1-v);
float color += source(x+1,y) * (u-x) * (y+1-v);
float color += source(x,y+1) * (x+1-u) * (v-y);
float 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:**
  
  \[ Gau(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-u)^2+(y-v)^2}{2\sigma^2}} \]

- **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
float Gaussian(float x, float y, float u, float v, float sigma) {
    float w = Gaussian(x, y, u, v, sigma);
    color += source(x, y) * w;
    sum += w;
}
```

**Result Comparison**