Lecture 20: Rotating, Scaling, Shifting and Shearing an Image

ECE 417: Multimedia Signal Processing
Mark Hasegawa-Johnson

University of Illinois

Nov. 1, 2018
1. Modifying an Image by Moving Its Points

2. Image Interpolation

3. Affine Transformations

4. Conclusions
<table>
<thead>
<tr>
<th></th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Modifying an Image by Moving Its Points</td>
</tr>
<tr>
<td>2</td>
<td>Image Interpolation</td>
</tr>
<tr>
<td>3</td>
<td>Affine Transformations</td>
</tr>
<tr>
<td>4</td>
<td>Conclusions</td>
</tr>
</tbody>
</table>
Moving Points Around

First, let’s suppose that somebody has given you a bunch of points:
...and let’s suppose you want to move them around, to create new images...
Moving One Point

- Your goal is to synthesize an output image, \( J[x, y] \), where \( J[x, y] \) might be intensity, or RGB vector, or whatever, \( x \) is row number (measured from top to bottom), \( y \) is column number (measured from left to right).

- What you have available is:
  - An input image, \( I[m, n] \), sampled at integer values of \( m \) and \( n \).
  - Knowledge that the input point at \( I(u, v) \) has been moved to the output point at \( J[x, y] \), where \( x \) and \( y \) are integers, but \( u \) and \( v \) might not be integers.

\[
J[x, y] = I(u, v)
\]
**Integer Output Points**

You want to create the output image as

\[
\text{for } x \text{ in range}(0, M): \\
\text{for } y \text{ in range}(0, N): \\
(u, v) = \text{input\_pixels\_corresponding\_to}(x, y) \\
J[x, y] = \text{compute\_pixel}(I, u, v)
\]

**Non-Integer Input Points**

If \([x, y]\) are integers, then usually, \((u, v)\) are not integers.
The function `compute_pixel` performs image interpolation. Given the pixels of $I[m, n]$ at integer values of $m$ and $n$, it computes the pixel at a non-integer position $I(u, v)$ as:

$$I(u, v) = \sum_{m} \sum_{n} I[m, n] h(u - m, v - n)$$
Piece-Wise Constant Interpolation

\[ l(u, v) = \sum_m \sum_n l[m, n] h(u - m, v - n) \]  \hspace{1cm} (1)

For example, suppose

\[ h(u, v) = \begin{cases} 
1 & 0 \leq u < 1, \ 0 \leq v < 1 \\
0 & \text{otherwise}
\end{cases} \]

Then Eq. (1) is the same as just truncating \( u \) and \( v \) to the next-lower integer, and outputting that number:

\[ l(u, v) = l[\lfloor u \rfloor, \lfloor v \rfloor] \]

where \( \lfloor u \rfloor \) means “the largest integer smaller than \( u \)”. 
Bi-Linear Interpolation

\[ l(u, v) = \sum_m \sum_n l[m, n] h(u - m, v - n) \]

For example, suppose

\[ h(u, v) = \max(0, (1 - |u|)(1 - |v|)) \]

Then Eq. (1) is the same as piece-wise linear interpolation among the four nearest pixels. This is called **bilinear interpolation** because it’s linear in two directions.

\[
\begin{align*}
    m &= \lfloor u \rfloor, \quad e = u - m \\
    n &= \lfloor v \rfloor, \quad f = v - m \\
    l(u, v) &= (1 - e)(1 - f)l[m, n] + (1 - e)fl[m, n + 1] \\
    &\quad + e(1 - f)l[m + 1, n] + ef[l[m + 1, n + 1]
\end{align*}
\]
Sinc Interpolation

\[ I(u, v) = \sum_m \sum_n I[m, n] h(u - m, v - n) \]

For example, suppose

\[ h(u, v) = \text{sinc}(\pi u)\text{sinc}(\pi v) \]

Then Eq. (1) is an ideal band-limited sinc interpolation. It guarantees that the continuous-space image, \( I(u, v) \), is exactly a band-limited D/A reconstruction of the digital image \( I[m, n] \).
Outline

1. Modifying an Image by Moving Its Points
2. Image Interpolation
3. Affine Transformations
4. Conclusions
How do we find \((u, v)\)?

Now the question: how do we find \((u, v)\)?

We’re going to assume that this is a piece-wise affine transformation.

\[
\begin{bmatrix}
  u \\
  v
\end{bmatrix} = \begin{bmatrix}
  a & b \\
  d & e
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix} + \begin{bmatrix}
  c \\
  f
\end{bmatrix}
\]
How do we find \((u, v)\)?

An affine transformation is defined by:

\[
\begin{bmatrix}
  u \\
  v
\end{bmatrix} = \begin{bmatrix}
  a & b & c \\
  d & e & f
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix} + \begin{bmatrix}
  c \\
  f
\end{bmatrix}
\]

A much easier to write this is by using extended-vector notation:

\[
\begin{bmatrix}
  u \\
  v \\
  1
\end{bmatrix} = \begin{bmatrix}
  a & b & c \\
  d & e & f \\
  0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

It’s convenient to define \(\vec{u} = [u, v, 1]^T\), and \(\vec{x} = [x, y, 1]^T\), so that for any \(\vec{x}\) in the output image,

\[
\vec{u} = A\vec{x}
\]
Notice that the affine transformation has 6 degrees of freedom: 
\((a, b, c, d, e, f)\). Therefore, you can accomplish 6 different types of transformation:

- Shift the image left ↔ right (using \(f\))
- Shift the image up ↔ down (using \(c\))
- Scale the image horizontally (using \(e\))
- Scale the image vertically (using \(a\))
- Rotate the image (using \(a, b, d, e\))
- Shear the image horizontally (using \(d\))

Vertical shear (using \(b\)) is a combination of horizontal shear + rotation.
Example: Reflection

\[
\begin{bmatrix}
\mathbf{u} \\
\mathbf{v} \\
1
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 \\
0 & -1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]
Example: Scale

\[
\begin{bmatrix}
u \\
v \\
1
\end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\]
Example: Rotation

\[
\begin{bmatrix}
u \\ v \\ 1
\end{bmatrix}
= \begin{bmatrix}
\cos \theta & \sin \theta & 0 \\
-\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\ y \\ 1
\end{bmatrix}
\]
Example: Shear

\[
\begin{bmatrix}
  u \\
  v \\
  1
\end{bmatrix} = \begin{bmatrix}
  1 & 0 & 0 \\
  0.5 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]
Affine Transformations

* Combines linear transformations, and Translations

\[
\begin{bmatrix}
    x' \\
    y' \\
    w
\end{bmatrix} =
\begin{bmatrix}
    a & b & c \\
    d & e & f \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    w
\end{bmatrix}
\]

the ones we looked at, that were the ones you know the rotation scaling and
Outline

1. Modifying an Image by Moving Its Points
2. Image Interpolation
3. Affine Transformations
4. Conclusions
You can generate an output image \( J[x, y] \) by warping an input image \( I(u, v) \).

If \((u, v)\) are not integers, you can compute the value of \( I(u, v) \) by interpolating among \( I[m, n] \), where \([m, n]\) are integers.

Shift, scale, rotation and shear are affine transformations, given by

\[
\begin{bmatrix}
    u \\
    v \\
    1
\end{bmatrix} =
\begin{bmatrix}
    a & b & c \\
    d & e & f \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    1
\end{bmatrix}
\]