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

Mark Hasegawa-Johnson

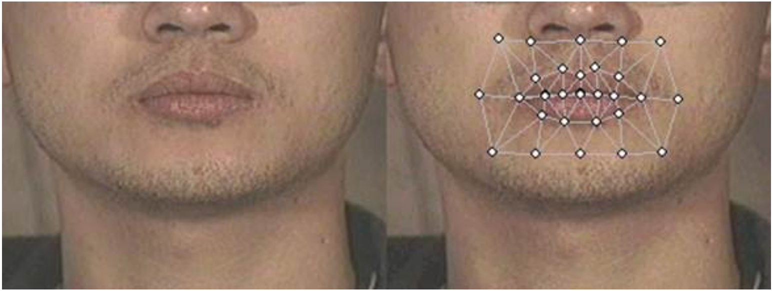
University of Illinois

ECE 417: Multimedia Signal Processing, Fall 2020

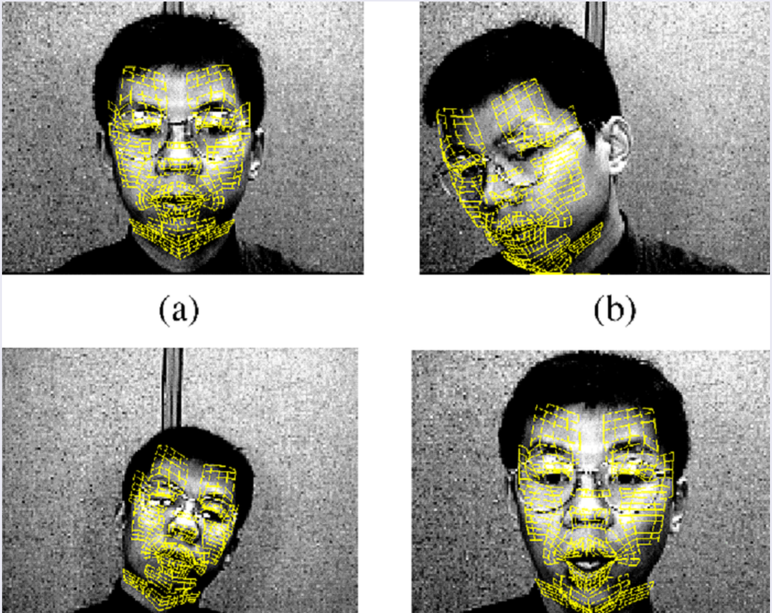
Outline

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, $J[y,x]$, where $J[y,x]$ might be intensity, or RGB vector, or whatever, y is **row** number (measured from top to bottom), x is **column** number (measured from left to right).
- What you have available is:
  - An input image, $I[n,m]$, sampled at integer values of m and n.
  - Knowledge that the input point at $I(v,u)$ has been **moved** to the output point at $J[y,x]$, where x and y are integers, but u and v might not be integers.

$$J[y,x] = I(v,u)$$

Outline

How do we find (u, v)?

Now the question: how do we find (u, v)?
For today, let's 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:

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

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,

$$\vec{u} = A\vec{x}$$

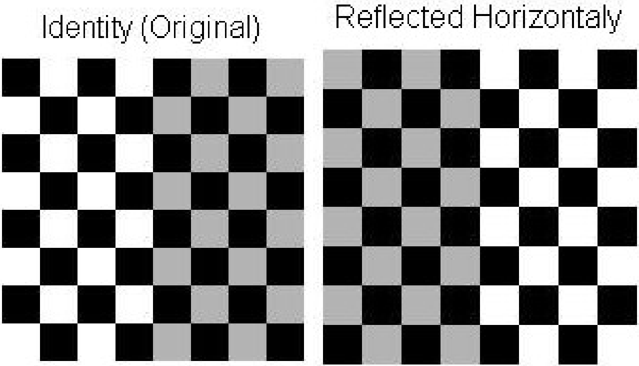
Affine Transforms

Notice that the affine transformation has 6 degrees of freedom: (a, b, c, d, e, f). Therefore, you can accmplish 6 different types of transformation:

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

Vertical shear (using d) is a combination of horizontal shear + rotation.

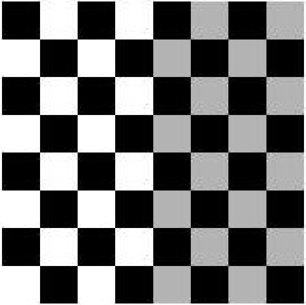
Example: Reflection



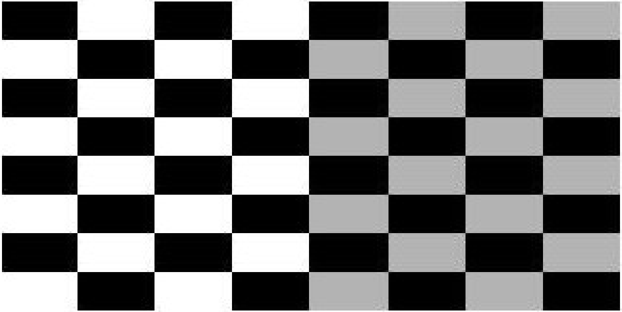
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Example: Scale

Identity (Original)

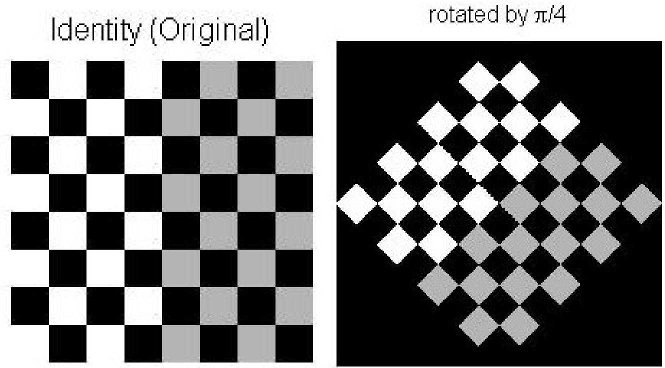


Scaled 2x Horizontaly



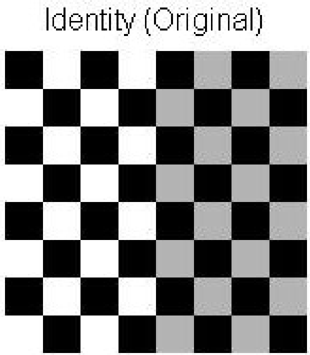
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

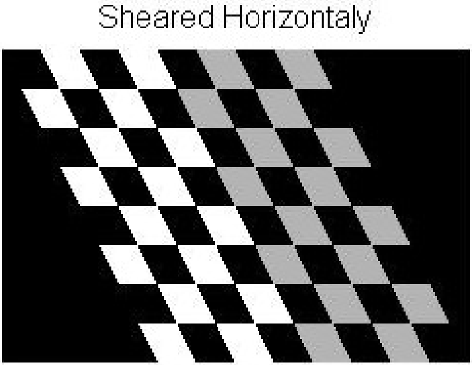
Example: Rotation



$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Example: Shear





$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$


Affine Transformations
Affine Transformations
* Combines linear transformations, and Translations
TECH
TECH
$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$
the ones we looked at, that were
the you know the rotation scaling and
0:26 / 1:19
HD

Outline

1. Modifying an Image by Moving Its Points
2. Affine Transformations
3. Image Interpolation
4. Conclusions

Integer Output Points

Now let's suppose that you've figured out the coordinate transform: for any given $J[y,x]$, you've figured out which pixel should be used to create it ($J[y,x] = I(v,u)$).

```
for x in range(0,M):
  for y in range(0,N):
    (u,v) = input_pixels_corresponding_to(x,y)
    J[y,x] = compute_pixel(I,v,u)
```

The Problem: Non-Integer Input Points

If $[x,y]$ are integers, then usually, (u,v) are not integers.

Image Interpolation

The function compute_pixel performs image interpolation. Given the pixels of $I[n, m]$ at integer values of m and n, it computes the pixel at a non-integer position $I(v, u)$ as:

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

Piece-Wise Constant Interpolation

$$I(v,u) = \sum_m \sum_n I[n,m]h(v-n, u-m) \qquad (1)$$

For example, suppose

$$h(v,u) = \begin{cases} 1 & 0 \le u < 1, \quad 0 \le v < 1 \\ 0 & \text{otherwise} \end{cases}$$

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

$$I(v,u) = I\left[\lfloor v \rfloor, \lfloor u \rfloor\right]$$

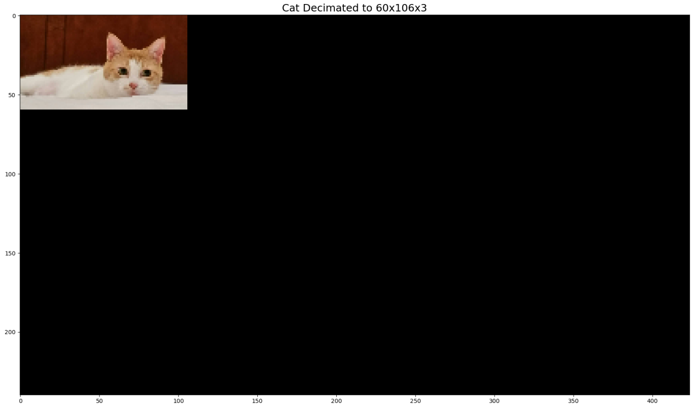
where $\lfloor u \rfloor$ means "the largest integer smaller than u".

Example: Original Image

For example, let's downsample this image, and then try to recover it by image interpolation:

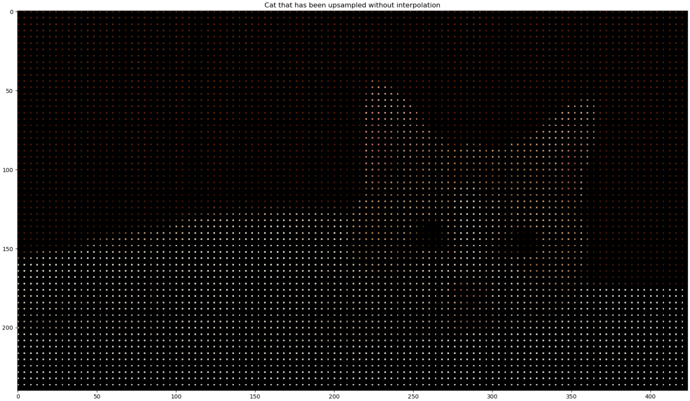
Example: Downsampled Image

Here's the downsampled image:



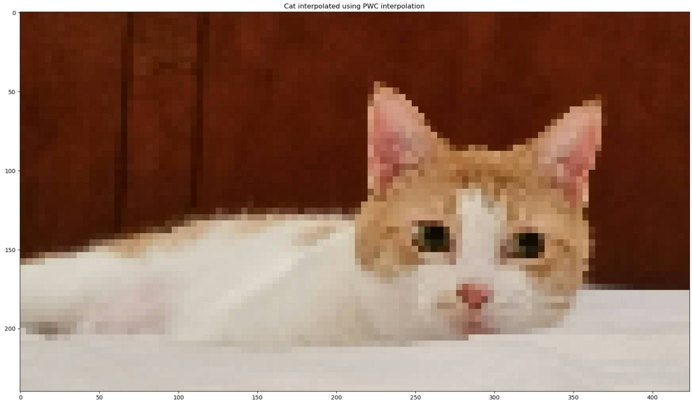
Example: Upsampled Image

Here it is after we upsample it back to the original resolution (insert 3 zeros between every pair of nonzero columns):



Example: PWC Interpolation

Here is the piece-wise constant interpolated image:



Bi-Linear Interpolation

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

For example, suppose

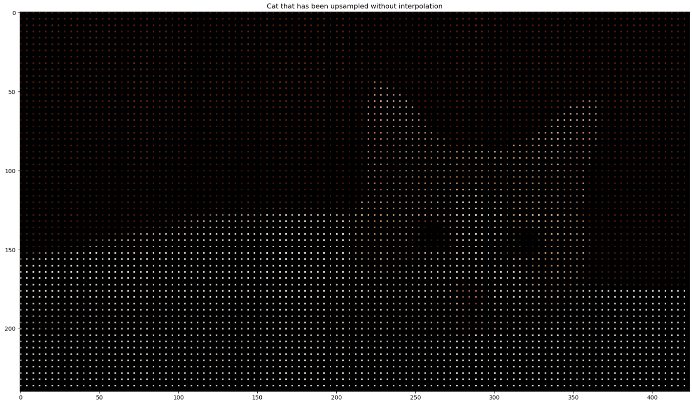
$$h(v,u) = \max\left(0, (1-|u|)(1-|v|)\right)$$

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.

$$\begin{aligned} m &= \lfloor u \rfloor, \quad e = u - m \\ n &= \lfloor v \rfloor, \quad f = v - m \\ I(v,u) &= (1-e)(1-f)I[n,m] + (1-e)fI[n,m+1] \\ &\quad + e(1-f)I[n+1,m] + efI[n+1,m+1] \end{aligned}$$

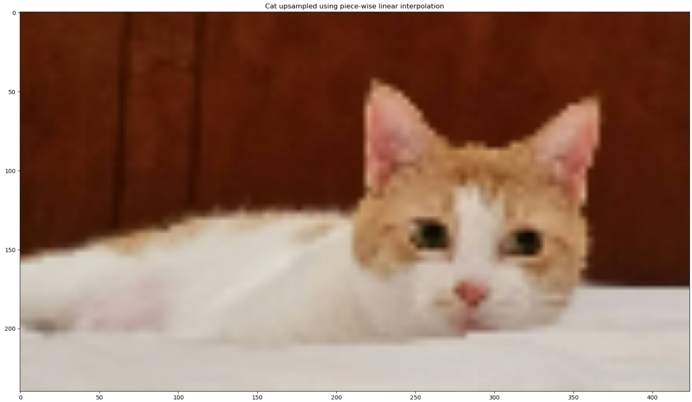
Example: Upsampled Image

Here's the upsampled image again:



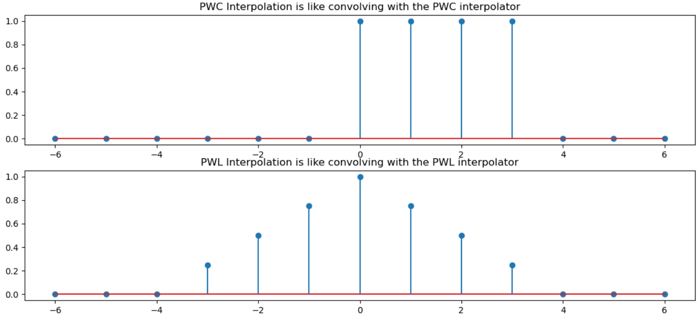
Example: Bilinear Interpolation

Here it is after bilinear interpolation:



PWC and PWL Interpolator Kernels

Bilinear interpolation uses a PWL interpolation kernel, which does not have the abrupt discontiuity of the PWC interpolator kernel.



Sinc Interpolation

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

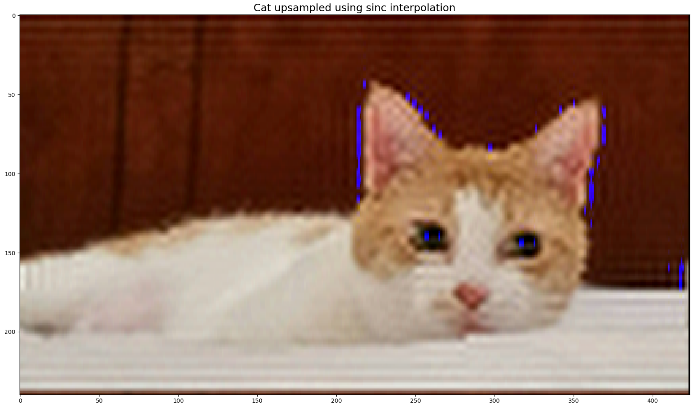
For example, suppose

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

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

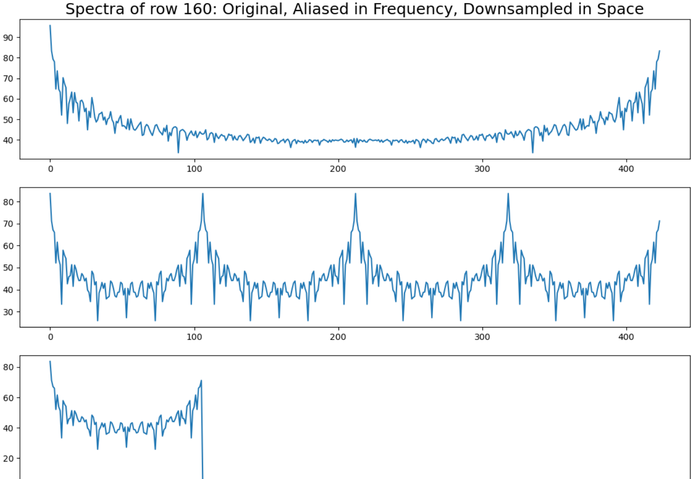
Sinc Interpolation

Here is the cat after sinc interpolation:



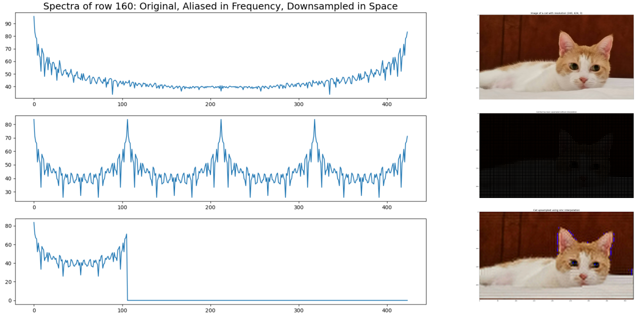
Original, Upsampled, and Sinc-Interpolated Spectra

Here are the magnitude Fourier transforms of the original, upsampled, and sinc-interpolated cat.



Original, Upsampled, and Sinc-Interpolated Spectra

Here are the magnitude Fourier transforms of the original, upsampled, and sinc-interpolated cat.



- The zeros in the upsampled cat correspond to aliasing in its spectrum.
- The ringing in the sinc-interpolated cat corresponds to the sharp cutoff, at pi/4, of its spectrum.

Outline

1. Modifying an Image by Moving Its Points
2. Affine Transformations
3. Image Interpolation
4. Conclusions

Conclusions

- You can generate an output image $J[y,x]$ by warping an input image $I(v,u)$.
- If (v,u) are not integers, you can compute the value of $I(v,u)$ by interpolating among $I[n,m]$, where $[n,m]$ are integers.

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

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

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