Lecture 3: Barycentric Coordinates and Image Interpolation

Mark Hasegawa-Johnson

University of Illinois

ECE 417: Multimedia Signal Processing, Fall 2023

Application: Animating a still image

Steps 2 and 3: Draw and Move Triangles

Step 4: Find the mapping between original and moved pixels: Barycentric coordinates

Step 5: Find the color of the source pixel: Bilinear interpolation

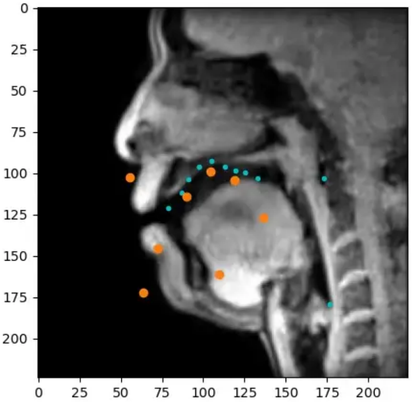
Conclusion

Outline

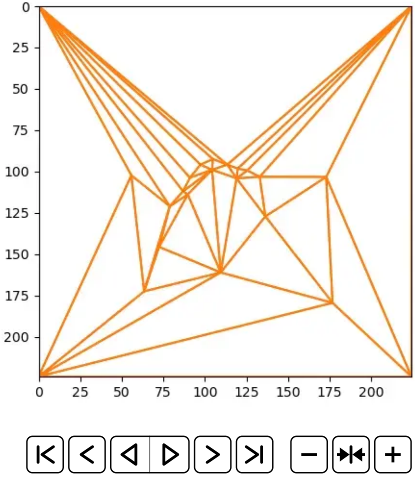
Strategy

1. Use affine projection to rotate, scale, and shear the XRMB points so that they match the shape of the MRI as well as possible.
2. Draw triangles on the MRI so that every pixel is inside a triangle.
3. Move the triangles.
4. Map each integer pixel in the target image to a real-valued pixel in the original image
5. Use bilinear interpolation to find the color of the pixel

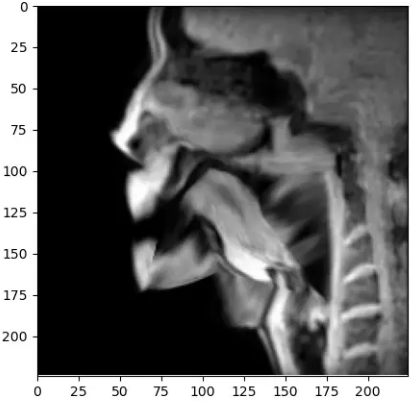
Step 1 (Last time): Use affine projection to map XRMB to MRI



Steps 2 and 3: Draw triangles, then move them



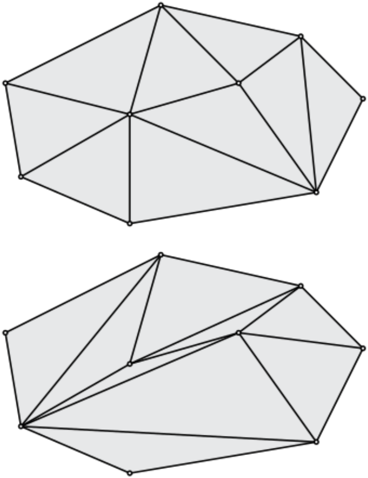
Steps 4 and 5: Find the color of each pixel



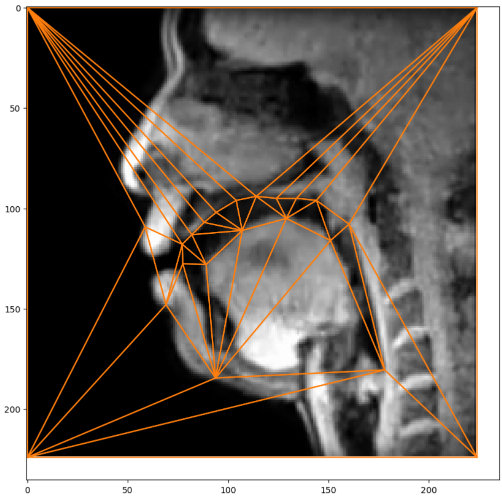
Outline

Triangulation

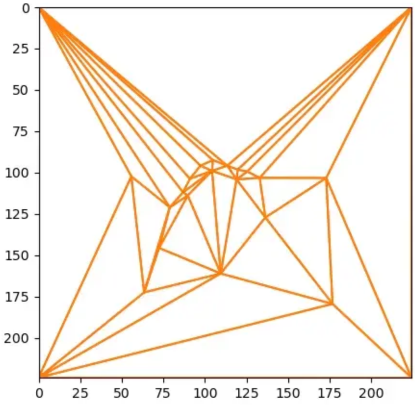
A **triangulation** of a set of points = a set of triangles, connecting those points, that covers the convex hull of those points.
There are many very cool algorithms that will automatically triangulate a set of points.



Triangulation for the machine problem is provided for you



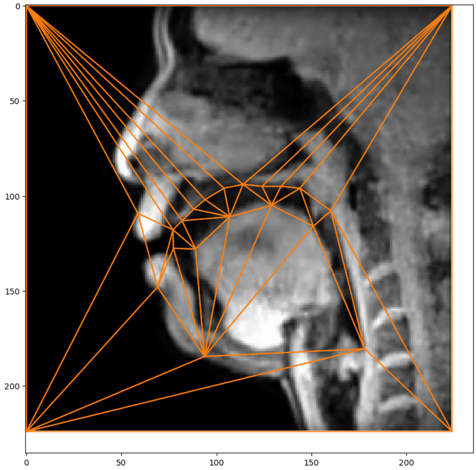
Once you have drawn the triangles, they move along with the points



Outline

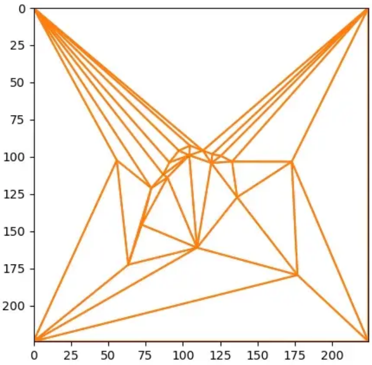
Source image

The source image is divided into non-overlapping triangles, X_k.



Target image

In the target image, those triangles have moved to new locations, Y_k.



Problem Statement: Moving pixels

- The size of the image we want to construct is $m \times n$.
- Consider a particular augmented target pixel, $y = \begin{bmatrix} y_1 \\ y_2 \\ 1 \end{bmatrix}$, where $0 \le y_1 \le m-1$ and $0 \le y_2 \le n-1$ are both integers.
- In order to find the color of target pixel y, we want to find out which source pixel, x, was moved to that location. Assume that pixels only move around – they don't change color while they move.

Problem Statement: Piece-wise affine transform

- Suppose that $y \in Y_k$, the k^{th} target triangle.
- We know therefore that $x \in X_k$. But there are lots of pixels inside X_k. Which one is it?
- Let's assume that, within each triangle, the pixels move according to an affine transform. In other words, if $y \in Y_k$, and if we already knew A_k, then we could find x by solving:

$$y = A_k x$$

where

$$y = \begin{bmatrix} y_1 \\ y_2 \\ 1 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}, \quad A_k = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ 0 & 0 & 1 \end{bmatrix}$$

Piece-wise affine transform

$$\text{target point: } y = \begin{bmatrix} y_1 \\ y_2 \\ 1 \end{bmatrix}, \quad \text{source point: } x = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}$$

Definition: if y is in the k^{th} triangle in the **output image**, then we want to use the k^{th} affine transform:

$$y = A_k x, \quad x = A_k^{-1} y$$

If **it is known that** $x = A_k^{-1} y$ for some unknown affine transform matrix A_k,

then

the method of barycentric coordinates finds x

without ever finding A_k.

Barycentric Coordinates

Barycentric coordinates turns the problem on its head. Suppose y is in a triangle with corners at y_1, y_2, and y_3. That means that

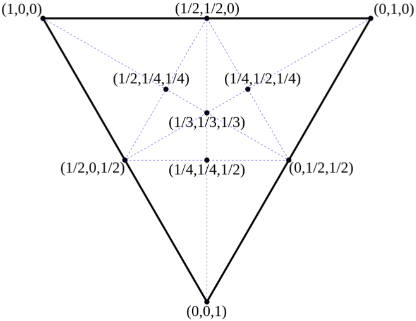
$$y = \beta_1 y_1 + \beta_2 y_2 + \beta_3 y_3$$

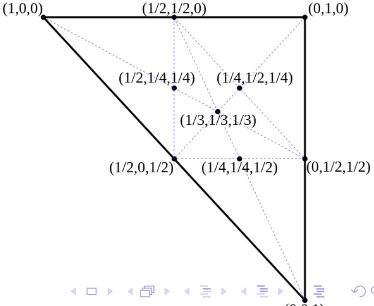
where

$$0 \leq \beta_1, \beta_2, \beta_3 \leq 1$$

and

$$\beta_1 + \beta_2 + \beta_3 = 1$$





Barycentric Coordinates

Suppose that all three of the corners are transformed by some affine transform A, thus

$$x_1 = Ay_1, \quad x_2 = Ay_2, \quad x_3 = Ay_3$$

Then if

$$\text{If: } y = \beta_1 y_1 + \beta_2 y_2 + \beta_3 y_3$$

Then:

$$\begin{aligned} x &= Ay \\ &= \beta_1 Ay_1 + \beta_2 Ay_2 + \beta_3 Ay_3 \\ &= \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \end{aligned}$$

In other words, once we know β, we no longer need to find A. We only need to know where the corners of the triangle have moved.

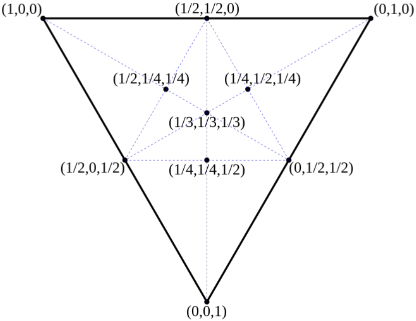
Barycentric Coordinates

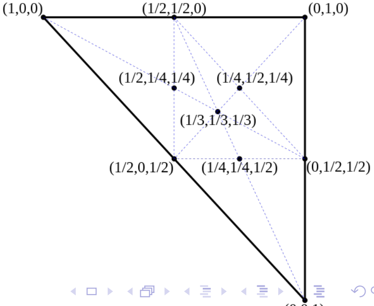
If

$$y = \beta_1 y_1 + \beta_2 y_2 + \beta_3 y_3$$

Then

$$x = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$





How to find Barycentric Coordinates

But how do you find β_1, β_2, and β_3?

$$\begin{bmatrix} y_1 \\ y_2 \\ 1 \end{bmatrix} = \beta_1 y_1 + \beta_2 y_2 + \beta_3 y_3 = \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} \\ y_{2,1} & y_{2,2} & y_{2,3} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}$$

Write this as:

$$y = Y\beta$$

Therefore

$$\beta = Y^{-1}y$$

This **always works:** the matrix Y is always invertible, unless all three of the points y_1, y_2, and y_3 are on a straight line.

How do you find out which triangle the point is in?

- Suppose we have K different triangles, each of which is characterized by a 3×3 matrix of its corners

$$Y_k = \left[y_1^{(k)}, y_2^{(k)}, y_3^{(k)}\right]$$

where $y_m^{(k)}$ is the m^{th} corner of the k^{th} triangle.

- Notice that, for any point y, for ANY triangle Y_k, we can find

$$\beta = Y_k^{-1} y$$

- However, the coefficients β_1, β_2, and β_3 will all be between 0 and 1 **if and only if** the point y is inside the triangle Y_k. Otherwise, some of the elements of β must be negative.

The Method of Barycentric Coordinates

To construct the animated output image frame $J[y_2, y_1]$, we do the following things:

- First, for each of the reference triangles X_k in the input image $I(x_2, x_1)$, decide where that triangle should move to. Call the new triangle location Y_k.
- Second, for each integer output pixel (y_1, y_2):
  - For each of the triangles, find $\beta = Y_k^{-1} y$.
  - Choose the triangle for which all of the elements of β are $0 \leq \beta_m \leq 1$.
  - Find $x = X_k \beta$.
  - Find the color of pixel $I(x_2, x_1)$ in the input image.
  - Set $J[y_2, y_1] = I(x_2, x_1)$.

Outline

Integer Target Points

Now let's suppose that you've figured out the coordinate transform: for any given $J[y_2, y_1]$, you've figured out which pixel should be used to create it ($J[y_2, y_1] = I(x_2, x_1)$).

$$x = X_k \beta = X_k Y_k^{-1} y$$

The Problem: Non-Integer Input Points

If $[y_2, y_1]$ are integers, then usually, (x_2, x_1) are not integers.

Image Interpolation

It is necessary to find $I(v, u)$ at non-integer values of (v, u) by interpolating between the integer-valued pixels provided in the image file.

Given the pixels of $I[n, m]$ at integer values of m and n, we can interpolate using an interpolation kernel $h(v, u)$:

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

Piece-Wise Constant Interpolation

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

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:

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

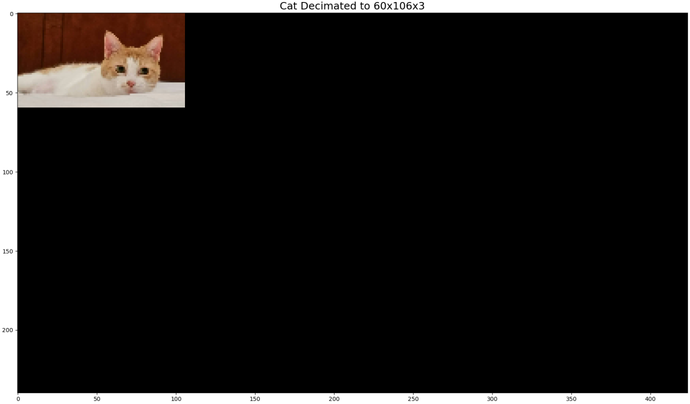
where $\lfloor u \rfloor$ means "the largest integer smaller than u".

Example: Original Image

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

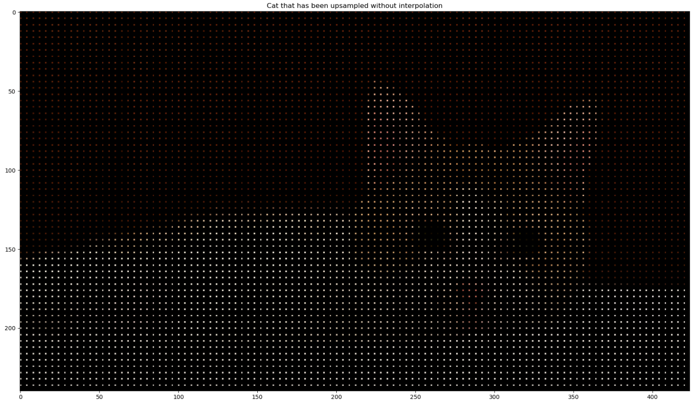
Example: Downsampled Image

Here's the downsampled image:



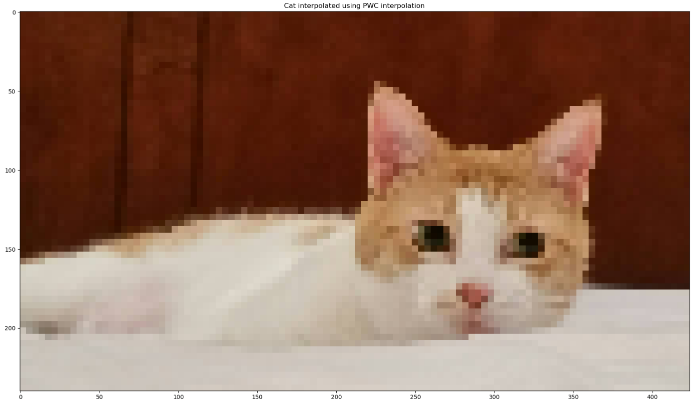
Example: Upsampled Image

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

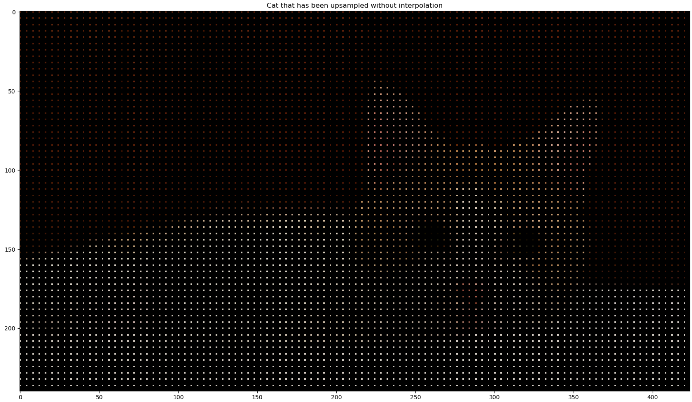
$$h(v,u) = \max\left(0, (1-|u|)(1-|v|)\right)$$

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.

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

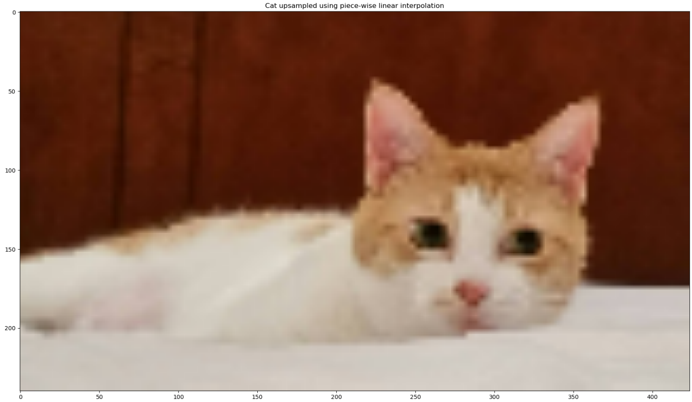
Example: Upsampled Image

Here's the upsampled image again:



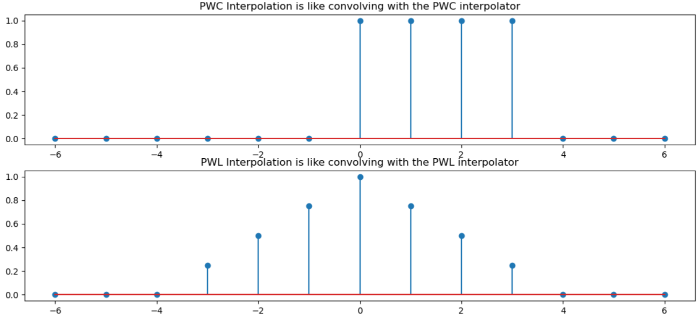
Example: Bilinear Interpolation

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

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

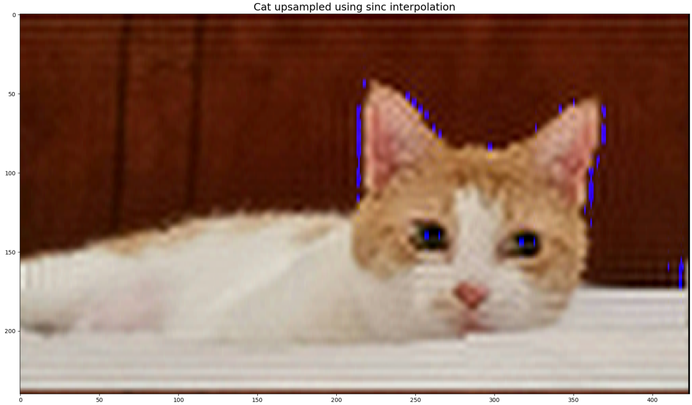
For example, suppose

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

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:



Summary of interpolation methods

- PWC interpolation results in a blocky image
- Sinc interpolation results in a smooth image, and would be perfect if the input image was infinite in size, but since real-world images have edges, the sinc interpolation produces ripple artifacts
- Bilinear interpolation is a very efficient solution with good results
- Better results are available using deep-learning-based super-resolution neural nets, but only after the neural net has been trained for a few weeks!

Outline

Strategy

1. Use MMSE affine projection to rotate, scale, and shear the XRMB points so that they match the shape of the MRI as well as possible.
2. Draw triangles on the MRI so that every pixel is inside a triangle.
3. Move the triangles.
4. Map each integer pixel in the target image to a real-valued pixel in the original image using barycentric coordinates
5. Use bilinear interpolation to find the color of the pixel