Geometric Image Manipulation

Lecture #9

February 11, 2002

Image Manipulation: Context

- Now we have the background in sampling (& Fourier analysis) to consider simple image transformations.
- There are always to components:
  - *Geometric*: finding which point in a source image correspondes to each point in the target image (or *vis-a-versa*)
  - *Photometric*: computing the value of the target pixel

Image Manipulation

- Step 1: Filter the source image, based on the Nyquist rate of the target
- Step 2: Calculate geometric transformation
- Step 3: Interpolate filtered values
  - In general, the geometric mapping will not map integer position onto integer position
  - Three methods:
    - Source → Target
    - Target → Source
    - 2 Pass (Source → Target)

Image Transformations

- The simplest set of transformations are translation, rotation, and scale
  - These are called the (6 DOF) affine transformations
- In matrix form these are:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} s1 & 0 \\ 0 & s2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

scale

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

rotation

and...

Image Transformations (II)

Translation
(note the 2D homogeneous coordinates)

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

(6 DOF) Affine Transformations

- Of course, these can be combined into one generic matrix:

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

- What else can you do with this matrix? (hint: two more transformation types)
- How can you specify this matrix?

Specifying Affine Transformations

- There are six unknowns in the matrix (a through f)
- If you specify one point in the source image and a corresponding point in the target image, that yields two equations:

$$u_i = ax_i + by_i + c$$
$$v_i = dx_i + ey_i + f$$

- So providing three point-to-point correspondences specifies an affine matrix

© Bruce A. Draper

Solving Affine Transformations

These linear equations can be easily solved:

- WLOG, assume $x_1=y_1=0$
- then u_1= c and v_1 = f
- so:

$$u_2 = ax_2 + by_2 + u_1$$
$$u_3 = ax_3 + by_3 + u_1$$
$$a = \frac{u_2 - u_1 - by_2}{x_2}$$
$$\frac{x_3(u_2 - u_1 - by_2)}{x_2} = u_3 - u_1 - by_3$$
$$\left[\frac{-x_3y_2}{x_2} - y_3\right]b = u_3 - u_1 - \frac{x_3}{x_2}(u_2 - u_1)$$
$$b = \frac{u_3 - u_1 - \frac{x_3}{x_2}(u_2 - u_1)}{\frac{-x_3y_2}{x_2} - y_3} = \frac{x_2(u_3 - u_2) - x_3(u_2 - u_1)}{-x_3y_2 - y_3x_2}$$

© Bruce A. Draper

Solving Affine (cont.)

- This can be substituted in to solve for a
- The same process with y's solves for d,e,f
- About the WLOG:
  - It was true because you can translate the original coordinate system by (-x1, -y_1)
  - So what do you do to compensate?
- Alternatively, set up a system of linear equations and solve...

© Bruce A. Draper

Perspective Transformations

- We can simulate more than just affine transformations
- We can do any perspective transformation of a <u>plane</u> to a <u>plane</u>.
- Therefore we can model an image as a plane in space, and project it onto any other image.
  - How does this differ from the perspective projection pipeline in CS410?

© Bruce A. Draper

Perspective Matrix

$$\begin{bmatrix} u' \\ v' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
$$u = {u'}/{w}, v = {v'}/{w}$$

- Why does element [3,3] = 1?
- How many points are needed to specify this matrix?

© Bruce A. Draper

Solving for Perspective

- Four corresponding points produce eight equations, eight unknowns --- but we can't observe *w*

$$u_i = {u'_i}/{w_i} = \frac{ax_i + by_i + c}{gx_i + hy_i + 1}$$
$$v_i = {v'_i}/{w_i} = \frac{dx_i + ey_i + f}{gx_i + hy_i + 1}$$

© Bruce A. Draper

Solving (cont.)

- Multiply to get rid of the fraction…

$$u_i\left(gx_i + hy_i + 1\right) = ax_i + by_i + c$$
$$v_i\left(gx_i + hy_i + 1\right) = dx_i + ey_i + f$$

- Now, remember that the u's,v's,x's & y's are known; group the unknown terms

$$u_i = ax_i + by_i + c - gx_iu_i - hy_iu_i$$
$$v_i = dx_i + ey_i + f - gx_iv_i - hy_iv_i$$

Solving (III)

- And express the result as a system of linear equations

$$\begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -y_2u_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2v_2 & -y_2v_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3u_3 & -y_3u_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3v_3 & -y_3v_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4u_4 & -y_4u_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4v_4 & -y_4v_4 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}$$

Solving (IV)

- Finally, invert the constant matrix and solve

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -y_2u_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2v_2 & -y_2v_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3u_3 & -y_3u_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3v_3 & -y_3v_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4u_4 & -y_4u_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4v_4 & -y_4v_4 \end{bmatrix}^{-1} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}$$

Intuitions for Perspective Image Transforms

- What does the following matrix do?

$$\begin{bmatrix} \sqrt{2} & -\sqrt{2} & 0 \\ \sqrt{2} & \sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

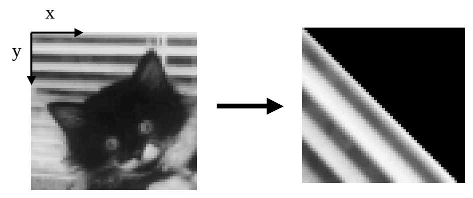
Matrix Decomposition

$$\underbrace{\begin{bmatrix} \sqrt{2} & -\sqrt{2} & 0 \\ \sqrt{2} & \sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{original}} = \underbrace{\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Scale by 2}} \underbrace{\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Rotation by 45}}$$

- Note that such decompositions are:
  - not unique (why?)
  - difficult to intuit

Intuitions...

This is the result of applying the matrix above...



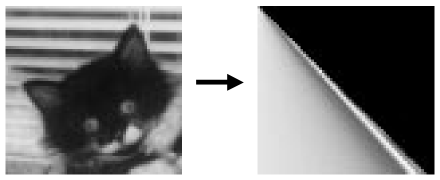
• Orientation of rotations is from positive X toward positive Y
• All orientations are about the origin!

More Intuitions

- What will the following matrix do?

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & -1 & 1 \end{bmatrix}$$

Check Your Intuitions



- What's going on here?

More Intuition Checking

- Part of what your seeing is a scale effect
  - positive terms in the bottom row create larger w values, and therefore smaller u,v values
- Something much weirder is also going on:
  - what happens when y = x+1?
  - How do you interpret this geometrically?
  - Isn't the perspective transform linear?
- So how do you select transform matrices?

Review: (2D) Perspective Transform

- Recall the basic equation for the perspective transform

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

$$u = u'/w, v = v'/w$$

- The only practical way to specify an image transform is by providing four point correspondences

Computing Transformations

- Remember how to build a transformation from four point correspondences….

$$\begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -y_2u_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2v_2 & -y_2v_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3u_3 & -y_3u_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3v_3 & -y_3v_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4u_4 & -y_4u_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4v_4 & -y_4v_4 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}$$

Computing...

- So if we want the following mapping:
  (0,0)→(0,0), (0,144)→(0,144),
  (152,0)→(152,50), (152,144)→(152,94)

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 144 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 144 & 1 & 0 & -20736 \\ 152 & 0 & 1 & 0 & 0 & 0 & -23104 & 0 \\ 0 & 0 & 0 & 152 & 0 & 1 & -7600 & 0 \\ 152 & 144 & 1 & 0 & 0 & 0 & -23104 & -21888 \\ 0 & 0 & 0 & 152 & 144 & 1 & -14288 & -13536 \end{bmatrix}$$

…More Computing...

$$\underbrace{\begin{bmatrix} -.014 & -.023 & .007 & .023 & .014 & .022 & -.007 & -.022 \\ -.007 & 0 & .007 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -.002 & -.014 & .002 & .007 & .002 & .014 & -.002 & -.007 \\ -.007 & -.007 & .007 & .007 & .007 & 0 & -.007 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ .000 & -.000 & .000 & .000 & .000 & .000 & -.000 & .000 \\ -.000 & 0 & .000 & 0 & .000 & 0 & -.000 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 144 \\ 152 \\ 50 \\ 152 \\ 94 \end{bmatrix}}_{M^{-1}} = \underbrace{\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}}_{\text{u\&v vector}} = \begin{bmatrix} 3.274 \\ 0 \\ 0 \\ 1.077 \\ 1 \\ 0 \\ .01497 \\ 0 \end{bmatrix}$$

…yields

$$\begin{bmatrix} 3.274 & 0 & 0 \\ 1.077 & 1 & 0 \\ .01497 & 0 & 1 \end{bmatrix}$$

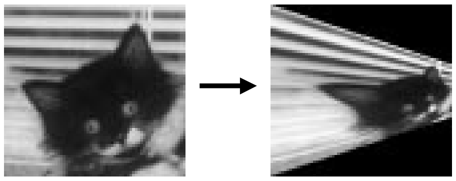


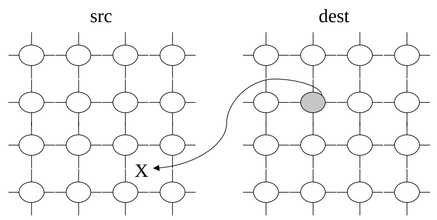
Image Manipulation (II)

- Step 1: Filter the source image, based on the Nyquist rate of the target
- Step 2: Calculate geometric transformation
  - Affine transformation, given three point correspondences
  - Perspective transformation, given four
- Step 3: Interpolate filtered values
  - Three methods:
    - Source $\rightarrow$ Target
    - Target $\rightarrow$ Source
    - 2 Pass (Source $\rightarrow$ Target)

Target $\rightarrow$ Source

- Invert transformation matrix computed on slide #15
- For every target pixel,
  - Apply (inverted) transform M to (x,y) coordinates
    - provides position of source data
    - In general, non-integer coordinates
  - If M(x,y) falls outside the source image, return black
  - Interpolate M(x,y) from filtered source pixel values
    - nearest neighbor (takes nearest source pixel)
    - bilinear
    - bicubic

Interpolation



Think of an image as a grid with pixels at the vertices. When applying a dest$\rightarrow$src transformation, the result will not fall exactly on a pixel (most of the time).

Bilinear Interpolation

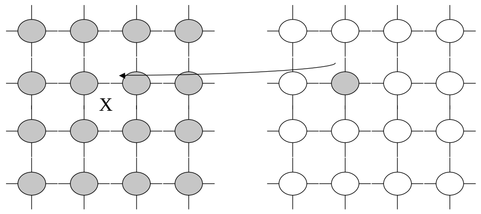
- Bilinear interpolation:
  - project target image to real-value source location
  - let tx = loc(x) - int(loc(x)), ty=loc(x)-int(loc(x))

$$p = \frac{P_{0,0}(1-tx)(1-ty) + P_{0,1}(1-tx)ty + P_{1,0}tx(1-ty) + P_{1,1}tx \cdot ty}{1}$$

- Good Points: identity transform does not smooth
- Bad Points: spatial block filter is horrible in frequency space
  - may cause frequency aliasing

Cubic Interpolation

Cubic interpolation uses the sixteen pixels around the source location for interpolation



Cubic Interpolation (II)

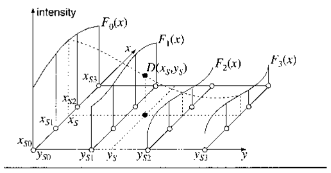
- Let x_{s0} = trunc(x_s)-1, $x_{s1} = x_{s0}+1$, $x_{s2} = x_{s0}+2$, $x_{s3}=x_{s0}+3$
- Let y_{s0} = trunc(y_s)-1, $y_{s1} = y_{s0}+1$, $y_{s2} = y_{s0}+2$, $y_{s3}=y_{s0}+3$

$$F_k(x) = a_k x^3 + b_k x^2 + c_k x + d_k$$
$$0 \le k \le 3$$
$$F_k(x_{sm}) = S(x_{sm}, y_{sk})$$

- Compute four cubic polynomials, one for each row:

Cubic Interpolation (III)

- Then compute one polynomial in y at x= x_s
- The value of this polynomial at y=y_s is the interpolated value.



From the Intel Image Processing Library Reference Manual, pg. B-6

Side Note:

Solving Cubic Equations

- Any cubic equation of the form:

$$y^3 + py^2 + qy + r = 0$$

- Can be rewritten as

$$x^3 + ax + b = 0$$

- By substituting:

$$y = x - \frac{p}{3}$$

- where:

$$a = \frac{1}{3}\left(3q - p^2\right),\ b = \frac{1}{27}\left(2p^3 - 9pq + 27r\right)$$

Solving Cubic Equations (II)

- Equations of the form:

$$x^3 + ax + b = 0$$

- Have a closed for solution. Let:

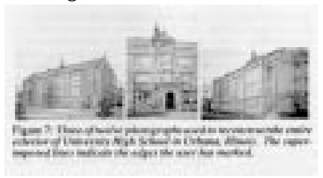
$$A = \sqrt[3]{-\frac{b}{2} + \sqrt{\frac{b^2}{4} + \frac{a^3}{27}}},\ B = \sqrt[3]{-\frac{b}{2} - \sqrt{\frac{b^2}{4} + \frac{a^3}{27}}}$$

- Then the roots are:

$$x = A + B,$$
$$x = -\frac{A+B}{2} + \frac{A-B}{2}\sqrt{-3}, x = -\frac{A+B}{2} - \frac{A-B}{2}\sqrt{-3}$$

Examples of Planar Transformations

- Given multiple images of different side of an object and a 3D model, you can "paint" the model with the images...



From Debevec, Taylor & Malik, SIGGRAPH '96

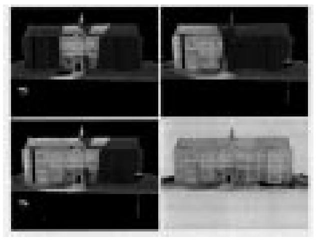
Examples (II)

…and then project the model to any other view

Figure 9: A synthetic view of University High School. This is a frame from an animation of flying around the entire building.

Examples (III)

Although it may require merging overlapping views



Transformations:

- So far, we have discussed only target→source transformations:
  - Guaranteed to leave no holes
  - Identity transform blurs image (unless NN interpolation)
  - May skip source pixels if shrinking source
- Alternative: source →target
  - For every source pixel,
    - project four corner points into target image
    - calculate overlap with target pixels (expensive)
    - treat target image as accumulator array
  - Warning: may leave holes if expanding source

2-Pass Transformations

- An alternative for Affine transformations is a 2 pass approach:
  - Any affine transformation can be broken down into
    - a linear transform in X, followed by
    - a not-neccessarily-linear transform in Y

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

2-Pass (II)

$$\left.\begin{bmatrix} u \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\right\}$$

This pass read the rows of the image, adjusting x

$$\left.\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ ? & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ y \\ 1 \end{bmatrix}\right\}$$

Now we want to read the columns of the image – but what value is ?

2-Pass (III)

Let ? be g:

$$\begin{aligned} v &= dx + ey + f \\ &= gu + ey + f \end{aligned} \qquad \begin{aligned} gu &= dx \\ u &= ax + by + c \\ x &= \frac{u - by - c}{a} \end{aligned}$$

Non linear, but easy to compute $\Big\{$

$$g = \frac{dx}{u} = \frac{d(u - by - c)}{ua}$$

2 Pass (IV)

- Why would you do this?
  - Very fast on vector hardware
    - stream the image through by rows, adjusting x
    - stream the image through by columns, adjusting y
  - Handy when morphing splines…
  - Linear in both directions for rotation