Many slides thanks to James Hays’ old CS 129 course, along with all of its acknowledgements.
Image Transformations

image filtering: change **range** of image

\[ g(x) = T(f(x)) \]

image warping: change **domain** of image

\[ g(x) = f(T(x)) \]
Image Transformations

image filtering: change **range** of image

\[ g(x) = T(f(x)) \]

image warping: change **domain** of image

\[ g(x) = f(T(x)) \]
Parametric (global) warping

Transformation $T$ is a coordinate-changing machine:

$$p' = T(p)$$

What does it mean that $T$ is global and parametric?

- Is the same for any point $p$
- Can be described by just a few numbers (parameters)

Let’s represent $T$ as a matrix:

$$p' = Mp$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
Think-Pair-Share – What happens?

Input image

1. \[
\begin{bmatrix}
    x' \\
y'
\end{bmatrix} = \begin{bmatrix}
    1 & 0 \\
    0 & 1
\end{bmatrix} \begin{bmatrix}
    x \\
y
\end{bmatrix}
\]

2. \[
\begin{bmatrix}
    x' \\
y'
\end{bmatrix} = \begin{bmatrix}
    2 & 0 \\
    0 & 2
\end{bmatrix} \begin{bmatrix}
    x \\
y
\end{bmatrix}
\]

3. \[
\begin{bmatrix}
    x' \\
y'
\end{bmatrix} = \begin{bmatrix}
    2 & 0 \\
    0 & 0.5
\end{bmatrix} \begin{bmatrix}
    x \\
y
\end{bmatrix}
\]

4. \[
\begin{bmatrix}
    x' \\
y'
\end{bmatrix} = \begin{bmatrix}
    -1 & 0 \\
    0 & 0.5
\end{bmatrix} \begin{bmatrix}
    x \\
y
\end{bmatrix}
\]

5. \[
\begin{bmatrix}
    x' \\
y'
\end{bmatrix} = \begin{bmatrix}
    1 & 0.1 \\
    0 & 1
\end{bmatrix} \begin{bmatrix}
    x \\
y
\end{bmatrix}
\]
Think-Pair-Share – What happens?

1. \[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  1 & 0 \\
  0 & 1
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

2. \[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  2 & 0 \\
  0 & 2
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

3. \[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  2 & 0 \\
  0 & 0.5
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

4. \[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  -1 & 0 \\
  0 & 0.5
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

5. \[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  1 & 0.1 \\
  0 & 1
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

Output image
2x2 Matrices

2D Mirror about Y axis

\[ x' = -x \]
\[ y' = y \]

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} =
\begin{bmatrix}
  -1 & 0 \\
  0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

2D Mirror over (0,0)

\[ x' = -x \]
\[ y' = -y \]

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} =
\begin{bmatrix}
  -1 & 0 \\
  0 & -1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]
Scaling

Scaling operation:  \[ x' = ax \]
\[ y' = by \]

Or, in matrix form:
\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix}
= \begin{bmatrix}
  a & 0 \\
  0 & b
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

scaling matrix \( S \)

Uniform scaling \( \Rightarrow a = b \)

What’s the inverse of \( S \)?
2-D Rotation

\[ x = r \cos(\phi) \]
\[ y = r \sin(\phi) \]
\[ x' = r \cos(\phi + \theta) \]
\[ y' = r \sin(\phi + \theta) \]

Trigonometric identity for angle sum
\[ x' = r \cos(\phi) \cos(\theta) - r \sin(\phi) \sin(\theta) \]
\[ y' = r \sin(\phi) \cos(\theta) + r \cos(\phi) \sin(\theta) \]

Substitute…
\[ x' = x \cos(\theta) - y \sin(\theta) \]
\[ y' = x \sin(\theta) + y \cos(\theta) \]
2-D Rotation

This is easy to capture in matrix form:

\[
\begin{bmatrix}
  x' \\
y'
\end{bmatrix} = \begin{bmatrix}
  \cos(\theta) & -\sin(\theta) \\
  \sin(\theta) & \cos(\theta)
\end{bmatrix} \begin{bmatrix}
  x \\
y
\end{bmatrix}
\]

Even though \( \sin(\theta) \) and \( \cos(\theta) \) are nonlinear functions of \( \theta \),
- \( x' \) is a linear combination of \( x \) and \( y \)
- \( y' \) is a linear combination of \( x \) and \( y \)

What is the inverse transformation?
- Rotation by \( -\theta \)
- For rotation matrices
  \[
  R^{-1} = R^T
  \]

Properties of rotation matrices:
- Square
- Determinant is 1
  \[
  \det = \cos(\theta)\cos(\theta) - (-\sin(\theta)\sin(\theta))
  \]
- Orthogonal basis (inverse is transpose). This means:
  - Dot product of any row or column with itself is 1.
  - Dot product of a row with any other row is 0.
  - Dot product of a column with any other column is 0.
2x2 Matrices

What types of transformations can be represented with a 2x2 matrix?

2D Identity?

\[
\begin{align*}
    x' &= x \\
y' &= y
\end{align*}
\]

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]

2D Scale around (0,0)?

\[
\begin{align*}
    x' &= s_x \cdot x \\
y' &= s_y \cdot y
\end{align*}
\]

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]
2x2 Matrices

What types of transformations can be represented with a 2x2 matrix?

2D Rotate around (0,0)?

\[ x' = \cos \Theta \cdot x - \sin \Theta \cdot y \]
\[ y' = \sin \Theta \cdot x + \cos \Theta \cdot y \]

\[
\begin{bmatrix}
    x' \\
    y'
\end{bmatrix}
= 
\begin{bmatrix}
    \cos \Theta & -\sin \Theta \\
    \sin \Theta & \cos \Theta
\end{bmatrix}
\begin{bmatrix}
    x \\
    y
\end{bmatrix}
\]

2D Shear?

\[ x' = x + s h_x \cdot y \]
\[ y' = s h_y \cdot x + y \]

\[
\begin{bmatrix}
    x' \\
    y'
\end{bmatrix}
= 
\begin{bmatrix}
    1 & s h_x \\
    s h_y & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y
\end{bmatrix}
\]
2x2 Matrices

What types of transformations can be represented with a 2x2 matrix?

2D Translation?

\[ x' = x + t_x \]
\[ y' = y + t_y \]

NO!

Only linear 2D transformations can be represented with a 2x2 matrix
Parametric (global) warping

Examples of parametric warps:

2x2 matrices

- Mirror
- Shear
- Scale / aspect
- Rotation
- perspective
- Translation
- cylindrical
All 2D Linear Transformations

Linear transformations are combinations of ...

- Scale,  
- Rotation,  
- Shear, and  
- Mirror

Properties of linear transformations:

- Origin maps to origin  
- Lines map to lines  
- Parallel lines remain parallel  
- Ratios are preserved  
- Closed under composition

\[
\begin{bmatrix}
{x'} \\
{y'}
\end{bmatrix} = \begin{bmatrix}
a & b \\
c & d
\end{bmatrix} \begin{bmatrix}
x \\
y
\end{bmatrix}
\]

\[
\begin{bmatrix}
{x'} \\
{y'}
\end{bmatrix} = \begin{bmatrix}
a & b & e & f \\
c & d & g & h \\
i & j & k & l
\end{bmatrix} \begin{bmatrix}
x \\
y
\end{bmatrix}
\]

- Linear transforms of homogeneous coordinates.
Homogeneous Coordinates

Q: How can we represent translation as a 3x3 matrix?

\[ x' = x + t_x \]
\[ y' = y + t_y \]
Homogeneous Coordinates

Represent coordinates in 2 dimensions with a 3-vector

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{Homogeneous coords} \quad \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
Homogeneous Coordinates

Add a 3rd coordinate to every 2D point

• \((x, y, w)\) represents a point at location \((x/w, y/w)\)
• \((x, y, 0)\) represents a point at infinity
• \((0, 0, 0)\) is not allowed

Convenient coordinate system to represent many useful transformations

\((2, 1, 1)\) or \((4, 2, 2)\) or \((6, 3, 3)\)
Homogeneous Coordinates

Q: How can we represent translation as a 3x3 matrix?

\[ x' = x + t_x \]
\[ y' = y + t_y \]

A: Using the rightmost column:

\[
\begin{bmatrix}
1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1
\end{bmatrix}
\]

Translation
Translation

Example of translation

Homogeneous Coordinates

\[
\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}
\]

$t_x = 2$

$t_y = 1$
Basic 2D Transformations

Basic 2D transformations as 3x3 matrices

Translate

\[
\begin{bmatrix}
  x' \\
y' \\
1
\end{bmatrix} = \begin{bmatrix}
  1 & 0 & t_x \\
  0 & 1 & t_y \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]

Scale

\[
\begin{bmatrix}
  x' \\
y' \\
1
\end{bmatrix} = \begin{bmatrix}
  s_x & 0 & 0 \\
  0 & s_y & 0 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]

Rotate

\[
\begin{bmatrix}
  x' \\
y' \\
1
\end{bmatrix} = \begin{bmatrix}
  \cos \Theta & -\sin \Theta & 0 \\
  \sin \Theta & \cos \Theta & 0 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]

Shear

\[
\begin{bmatrix}
  x' \\
y' \\
1
\end{bmatrix} = \begin{bmatrix}
  1 & sh_x & 0 \\
  sh_y & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]
Matrix Composition

Transformations can be combined by matrix multiplication because they are *linear!*

\[
\begin{bmatrix}
    x' \\
y' \\
w'
\end{bmatrix} = \begin{bmatrix}
    1 & 0 & tx \\
    0 & 1 & ty \\
    0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
    \cos \Theta & -\sin \Theta & 0 \\
    \sin \Theta & \cos \Theta & 0 \\
    0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
    sx & 0 & 0 \\
    0 & sy & 0 \\
    0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
x \\
y \\
w
\end{bmatrix}
\]

\[p' = T(t_x,t_y) R(\Theta) S(s_x,s_y) p\]
Parametric (global) warping

Examples of parametric warps:

2x2 matrices

Mirror
Shear
Scale / aspect
Rotation

3x3 homogeneous matrices

Translation
Affine transformations are combinations of …

- Linear transformations, and
- Translations

Properties of affine transformations:

- **Origin does not necessarily map to origin**
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition
- Models change of basis

Will the last coordinate $w$ always be 1?
Projective Transformsations

Projective transformations …

- Affine transformations, and
- Projective warps

Properties of projective transformations:

- Origin does not necessarily map to origin
- Lines map to lines
- Parallel lines do not necessarily remain parallel
- Ratios are not preserved
- Closed under composition
- Models change of basis

\[
\begin{bmatrix}
  x' \\
  y' \\
  w'
\end{bmatrix} =
\begin{bmatrix}
  a & b & c \\
  d & e & f \\
  g & h & i
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  w
\end{bmatrix}
\]
Parametric (global) warping

Examples of parametric warps:

2x2 matrices

- Mirror
- Shear
- Scale / aspect
- Rotation

3x3 matrices

- Translation
- perspective
2D image transformations

These transformations are a nested set of groups
- Closed under composition and inverse is a member

<table>
<thead>
<tr>
<th>Name</th>
<th>Matrix</th>
<th># D.O.F.</th>
<th>Preserves:</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>translation</td>
<td>$\begin{bmatrix} I &amp; t \end{bmatrix}_{2x3}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rigid (Euclidean)</td>
<td>$\begin{bmatrix} R &amp; t \end{bmatrix}_{2x3}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>similarity</td>
<td>$\begin{bmatrix} sR &amp; t \end{bmatrix}_{2x3}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>affine</td>
<td>$\begin{bmatrix} A \end{bmatrix}_{2x3}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>projective</td>
<td>$\begin{bmatrix} \tilde{H} \end{bmatrix}_{3x3}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Recovering Transformations

What if we know \( f \) and \( g \) and want to recover the transform \( T \)?

- e.g. better align images from Project 1
- willing to let user provide correspondences
  - How many do we need?
Translation: # correspondences?

How many correspondences needed for translation?

How many Degrees of Freedom?

What is the transformation matrix? \[
\begin{bmatrix}
1 & 0 & \ p'_x - \ p_x \\
0 & 1 & \ p'_y - \ p_y \\
0 & 0 & 1
\end{bmatrix}
\]
Euclidian: # correspondences?

How many correspondences needed for translation+rotation?
How many DOF?
Affine: # correspondences?

How many correspondences needed for affine? How many DOF?
How many correspondences needed for projective?
How many DOF?
Given a coordinate transform \((x', y') = T(x, y)\) and a source image \(f(x, y)\), how do we compute a transformed image \(g(x', y') = f(T(x, y))\)?
Forward warping

Send each pixel $f(x,y)$ to its corresponding location $(x',y') = T(x,y)$ in the second image.

Q: what if pixel lands “between” two pixels?
Forward warping

Send each pixel \( f(x,y) \) to its corresponding location \((x',y') = T(x,y)\) in the second image

Q: what if pixel lands “between” two pixels?
A: distribute color among neighboring pixels \((x',y')\)
   - Known as “splatting”
   - Check out `griddata` in Matlab
Inverse warping

Get each pixel $g(x',y')$ from its corresponding location

$$(x,y) = T^{-1}(x',y')$$ in the first image

Q: what if pixel comes from “between” two pixels?
Inverse warping

Get each pixel \( g(x',y') \) from its corresponding location \( (x,y) = T^{-1}(x',y') \) in the first image.

Q: what if pixel comes from “between” two pixels?
A: *Interpolate* color value from neighbors
   - nearest neighbor, bilinear, Gaussian, bicubic
   - Check out `interp2` in Matlab
Forward vs. inverse warping

Q: Which is better?

A: usually inverse—eliminates holes
   • however, it requires an invertible warp function—not always possible...
Pinhole camera model

\[ f = \text{Focal length} \]
\[ c = \text{Optical center of the camera} \]
Projection: world coordinates $\rightarrow$ image coordinates

Camera Center $(0, 0, 0)$

$P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$

$U = -X * \frac{f}{Z}$

$V = -Y * \frac{f}{Z}$

What is the effect if $f$ and $Z$ are equal?
Camera (projection) matrix

\[
x = K[R \ t]X
\]

- \( x \): Image Coordinates: \((u, v, 1)\)
- \( K \): Intrinsic Matrix (3x3)
- \( R \): Rotation (3x3)
- \( t \): Translation (3x1)
- \( X \): World Coordinates: \((X, Y, Z, 1)\)
Projective geometry

- 2D point in cartesian = \((x,y)\) coordinates
- 2D point in projective = \((x,y,w)\) coordinates
Projective geometry

- 2D point in cartesian = \((x,y)\) coordinates
- 2D point in projective = \((x,y,w)\) coordinates
Varying $w$

Projected image becomes smaller.

$W_1$

$W_2 < W_1$

Projected image becomes smaller.
Projective geometry

- 2D point in projective = \((x, y, w)\) coordinates
  - \(w\) defines the scale of the projected image.
  - Each \(x, y\) point becomes a ray!
Projective geometry

- In 3D, point \((x,y,z)\) becomes \((x,y,z,w)\)
- Perspective is \(w\) varying with \(z\):
  - Objects far away are appear smaller
Homogeneous coordinates

Converting to homogeneous coordinates

\[(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{2D (image) coordinates}\]

\[(x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{3D (scene) coordinates}\]

Converting from homogeneous coordinates

\[\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow \left(\frac{x}{w}, \frac{y}{w}\right) \quad \text{2D (image) coordinates}\]

\[\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right) \quad \text{3D (scene) coordinates}\]
Homogeneous coordinates

Scale invariance in projection space

\[
\begin{bmatrix}
    x \\
    y \\
    w
\end{bmatrix}
= 
\begin{bmatrix}
    kx \\
    ky \\
    kw
\end{bmatrix}
\Rightarrow

\begin{bmatrix}
    kx \\
    ky \\
    kw
\end{bmatrix}
= 
\begin{bmatrix}
    \frac{kx}{kw} \\
    \frac{ky}{kw} \\
    \frac{kw}{w}
\end{bmatrix}
\]

Homogeneous Coordinates \hspace{1cm} Cartesian Coordinates

E.G., we can uniformly scale the projective space, and it will still produce the same image -> scale ambiguity
Camera (projection) matrix

\[
x = K [R \ t] X
\]

- **x**: Image Coordinates: \((u,v,1)\)
- **K**: Intrinsic Matrix (3x3)
- **R**: Rotation (3x3)
- **t**: Translation (3x1)
- **X**: World Coordinates: \((X,Y,Z,1)\)
Projection matrix

Intrinsic Assumptions
- Unit aspect ratio
- Optical center at (0,0)
- No skew

Extrinsic Assumptions
- No rotation
- Camera at (0,0,0)

\[
x = K[I \ 0]X
\]

\[
\begin{bmatrix}
u \\
w \\
1
\end{bmatrix} = \begin{bmatrix}
f & 0 & 0 & 0 \\
0 & f & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix} \begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]
Projection: world coordinates $\rightarrow$ image coordinates

\[ \mathbf{P} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \]

\[ \begin{align*}
U &= -X \frac{f}{Z} \\
V &= -Y \frac{f}{Z}
\end{align*} \]

\[ \mathbf{p} = \begin{bmatrix} U \\ V \end{bmatrix} \]

\( \mathbf{p} \) = distance from image center

Image center \((u_0, v_0)\)
Remove assumption: known optical center

Intrinsic Assumptions
- Unit aspect ratio
- No skew

Extrinsic Assumptions
- No rotation
- Camera at (0,0,0)

\[
x = K[I \ 0]X
\]

\[
\begin{bmatrix}
u \\ w \\ v \\ 1
\end{bmatrix} =
\begin{bmatrix}
f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\ y \\ z \\ 1
\end{bmatrix}
\]
Remove assumption: equal aspect ratio

Intrinsic Assumptions
• No skew

Extrinsic Assumptions
• No rotation
• Camera at (0,0,0)

\[ \mathbf{x} = \mathbf{K}[\mathbf{I} \ 0] \mathbf{X} \]
Remove assumption: non-skewed pixels

Intrinsic Assumptions

Extrinsic Assumptions
- No rotation
- Camera at (0,0,0)

\[
x = K[I \ 0]X
\]

Note: different books use different notation for parameters

James Hays
Oriented and Translated Camera
Allow camera translation

Intrinsic Assumptions

Extrinsic Assumptions
• No rotation

\[
x = K [I \ t] X
\]

\[
\begin{bmatrix}
u \\ v \\ 1
\end{bmatrix} =
\begin{bmatrix}
f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\ y \\ z \\ 1
\end{bmatrix}
\]
3D Rotation of Points

Rotation around the coordinate axes, counter-clockwise:

\[
R_x(\alpha) = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos \alpha & -\sin \alpha \\
0 & \sin \alpha & \cos \alpha \\
\end{bmatrix}
\]

\[
R_y(\beta) = \begin{bmatrix}
\cos \beta & 0 & \sin \beta \\
0 & 1 & 0 \\
-\sin \beta & 0 & \cos \beta \\
\end{bmatrix}
\]

\[
R_z(\gamma) = \begin{bmatrix}
\cos \gamma & -\sin \gamma & 0 \\
\sin \gamma & \cos \gamma & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\]
Allow camera rotation

$$x = K[R\ t]X$$

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = 
\begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
Demo – Kyle Simek

• “Dissecting the Camera Matrix”
• Three-part blog series
  • http://ksimek.github.io/2012/08/14/decompose/
  • http://ksimek.github.io/2012/08/22/extrinsic/
  • http://ksimek.github.io/2013/08/13/intrinsic/

• “Perspective toy”
  • http://ksimek.github.io/perspective_camera_toy.html
Orthographic Projection

- Special case of perspective projection
  - Distance from the COP to the image plane is infinite
  
- Also called “parallel projection”
- What’s the projection matrix?

\[
\begin{bmatrix}
u \\
v \\
w
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
1 & & & & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
\]
Geometry for a simple stereo system

- First, assuming parallel optical axes, known camera parameters (i.e., calibrated cameras):

Two cameras, simultaneous views
Geometry for a simple stereo system

- Assume parallel optical axes, known camera parameters (i.e., calibrated cameras).

**What is expression for $Z$?**

Similar triangles $(p_l, P, p_r)$ and $(O_l, P, O_r)$:

\[
\frac{T + x_l - x_r}{Z - f} = \frac{T}{Z}
\]

\[
Z = f \frac{T}{x_r - x_l}
\]

disparity
Stereo correspondence constraints

• Given p in left image, where can corresponding point p’ be?
Stereo correspondence constraints
Geometry of two views constrains where the corresponding pixel for some image point in the first view must occur in the second view.

- It must be on the line carved out by a plane connecting the world point and optical centers.
Epipolar geometry

http://www.ai.sri.com/~luong/research/Meta3DViewer/EpipolarGeo.html
Epipolar geometry: terms

- **Baseline**: line joining the camera centers
- **Epipole**: point of intersection of baseline with image plane
- **Epipolar plane**: plane containing baseline and world point
- **Epipolar line**: intersection of epipolar plane with the image plane

- All epipolar lines intersect at the epipole
- An epipolar plane intersects the left and right image planes in epipolar lines

*Why is the epipolar constraint useful?*
Fundamental matrix

Let \( x \) be a point in left image, \( x' \) in right image

Epipolar relation
- \( x \) maps to epipolar line \( l' \)
- \( x' \) maps to epipolar line \( l \)

Epipolar mapping described by a 3x3 matrix \( F \):

\[
l' = Fx \\
l = F^Tx'
\]

It follows that:

\[
x'Fx = 0
\]
Fundamental matrix

This matrix $F$ is called
- the “Essential Matrix”
  - when image intrinsic parameters are known
- the “Fundamental Matrix”
  - more generally (uncalibrated case)

Can solve for $F$ from point correspondences
- Each $(x, x')$ pair gives one linear equation in entries of $F$

$$x'Fx = 0$$

- $F$ has 9 entries, but really only 7 degrees of freedom.
- With 8 points it is simple to solve for $F$, but it is also possible with 7. See Marc Pollefey’s notes for a nice tutorial
Stereo image rectification
Stereo image rectification

- Reproject image planes onto a common plane parallel to the line between camera centers

- Pixel motion is horizontal after this transformation

- Two homographies (3x3 transform), one for each input image reprojection

Rectification example
Potential matches for $x$ have to lie on the corresponding line $l'$. 

Potential matches for $x'$ have to lie on the corresponding line $l$. 

Summary: Key idea: Epipolar constraint
Big idea

As with high dynamic range imaging, we are compensating for the shortcomings of traditional cameras by capturing and fusing multiple images.

Today we will assume that correspondence between photos is known and we focus on finding and applying homographies.
Homographies and Panoramas

Slides from Steve Seitz, Rick Szeliski, Alexei Efros, Fredo Durand, and Kristin Grauman
Why Mosaic?

Are you getting the whole picture?
  • Compact Camera FOV = 50 x 35°
Why Mosaic?

Are you getting the whole picture?

• Compact Camera FOV = 50 x 35°
• Human FOV = 200 x 135°
Why Mosaic?

Are you getting the whole picture?

- Compact Camera FOV = 50 x 35°
- Human FOV = 200 x 135°
- Panoramic Mosaic = 360 x 180°
Panoramic Photos are old

Sydney, 1875

Beirut, late 1800’s
Panorama Capture
Mosaics: stitching images together

virtual wide-angle camera
How to do it?

Basic Procedure

• Take a sequence of images from the same position
  – Rotate the camera about its optical center
• Compute transformation between second image and first
• Transform the second image to overlap with the first
• Blend the two together to create a mosaic
• If there are more images, repeat

…but wait, why should this work at all?

• What about the 3D geometry of the scene?
• Why aren’t we using it?
Aligning images

Translations are not enough to align the images
A pencil of rays contains all views

Can generate any synthetic camera view if it has the same center of projection!
Image reprojection

The mosaic has a natural interpretation in 3D

- The images are reprojected onto a common plane
- The mosaic is formed on this plane
- Mosaic is a *synthetic wide-angle camera*
Basic question

- How to relate two images from the same camera center?
  - how to map a pixel from PP1 to PP2

Answer

- Cast a ray through each pixel in PP1
- Draw the pixel where that ray intersects PP2

But don’t we need to know the geometry of the two planes with respect to the eye?

Observation:
Rather than thinking of this as a 3D reprojection, think of it as a 2D image warp from one image to another.
Which t-form is the right one for warping PP1 into PP2?
e.g. translation, Euclidean, affine, projective

Translation: 2 unknowns
Affine: 6 unknowns
Perspective: 8 unknowns
Homography

A: Projective – mapping between any two PPs with the same center of projection
  • rectangle should map to arbitrary quadrilateral
  • parallel lines aren’t maintained
  • but must preserve straight lines
  • Equal to: project, rotate, reproject
called Homography

\[
\begin{bmatrix}
w x' \\
w y' \\
w \\
\end{bmatrix}
= \begin{bmatrix}
* & * & * \\
* & * & * \\
* & * & * \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1 \\
\end{bmatrix}
\]

To apply a homography \( H \)
  • Compute \( p' = Hp \) (regular matrix multiply)
  • Convert \( p' \) from homogeneous to image coordinates
Image warping with homographies

image plane in front

black area where no pixel maps to
To unwarp (rectify) an image

- Find the homography $H$ given a set of $p$ and $p'$ pairs
- How many correspondences are needed?
- Tricky to write $H$ analytically, but we can solve for it!
  - Find such $H$ that “best” transforms points $p$ into $p'$
  - Use least-squares!
To apply a given homography \( H \)

- Compute \( p' = Hp \) (regular matrix multiply)
- Convert \( p' \) from homogeneous to image coordinates

\[
\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\]
To compute the homography given pairs of corresponding points in the images, we need to set up an equation where the parameters of $H$ are the unknowns...
Least Squares Example

Say we have a set of data points \((X_1, X_1'), (X_2, X_2'), (X_3, X_3'),\) etc.
(e.g. person’s height vs. weight)

We want a nice compact formula (a line) to predict \(X’s\) from \(Xs:\)
\[
Xa + b = X'
\]
We want to find \(a\) and \(b\)

How many \((X,X')\) pairs do we need?
\[
X_1a + b = X_1'
X_2a + b = X_2'
\]

What if the data is noisy?
\[
\begin{bmatrix}
X_1 \\ X_2 \\ X_3 \\ \vdots
\end{bmatrix}
\begin{bmatrix}
a \\ b
\end{bmatrix}
=
\begin{bmatrix}
X_1' \\ X_2' \\ X_3' \\ \vdots
\end{bmatrix}
\]
\[
\min \|Ax - B\|^2
\]
overconstrained
Solving for homographies

\[ p' = Hp \]

\[
\begin{bmatrix}
wx' \\
wy' \\
w
\end{bmatrix} =
\begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]

Can set scale factor \(i=1\). So, there are 8 unknowns.
Set up a system of linear equations:
\[ Ah = b \]
where vector of unknowns \(h = [a,b,c,d,e,f,g,h]^T\)
Need at least 8 eqs, but the more the better…
Solve for \(h\). If overconstrained, solve using least-squares:
\[
\min \|Ah - b\|^2
\]

\text{>> help lmdivide}
Example system for finding homography

...destination coordinates of the four corners of a quadrilateral. Let the correspondence map \((u_k, v_k)^T\) to \((x_k, y_k)^T\) for vertices numbered cyclically \(k = 0, 1, 2, 3\). All coordinates are assumed to be real (finite). To compute the forward mapping matrix \(M_{sd}\), assuming that \(i = 1\), we have eight equations in the eight unknowns \(a\)-\(h\):

\[
x_k = \frac{au_k + bv_k + c}{gu_k + hv_k + 1} \quad \Rightarrow \quad u_k a + v_k b + c - u_k x_k g - v_k x_k h = x_k
\]

\[
y_k = \frac{du_k + ev_k + f}{gu_k + hv_k + 1} \quad \Rightarrow \quad u_k d + v_k e + f - u_k y_k g - v_k y_k h = y_k
\]

for \(k = 0, 1, 2, 3\). This can be rewritten as an \(8 \times 8\) system:

\[
\begin{pmatrix}
    u_0 & v_0 & 1 & 0 & 0 & 0 & -u_0 x_0 & -v_0 x_0 \\
    u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1 x_1 & -v_1 x_1 \\
    u_2 & v_2 & 1 & 0 & 0 & 0 & -u_2 x_2 & -v_2 x_2 \\
    u_3 & v_3 & 1 & 0 & 0 & 0 & -u_3 x_3 & -v_3 x_3 \\
    0 & 0 & 0 & u_0 & v_0 & 1 & -u_0 y_0 & -v_0 y_0 \\
    0 & 0 & 0 & u_1 & v_1 & 1 & -u_1 y_1 & -v_1 y_1 \\
    0 & 0 & 0 & u_2 & v_2 & 1 & -u_2 y_2 & -v_2 y_2 \\
    0 & 0 & 0 & u_3 & v_3 & 1 & -u_3 y_3 & -v_3 y_3
\end{pmatrix}
\begin{pmatrix}
    a \\
    b \\
    c \\
    d \\
    e \\
    f \\
    g \\
    h
\end{pmatrix}
=
\begin{pmatrix}
    x_0 \\
    x_1 \\
    x_2 \\
    x_3 \\
    y_0 \\
    y_1 \\
    y_2 \\
    y_3
\end{pmatrix}
\]

Source: Paul Heckbert, *Fundamentals of Texture Mapping and Image Warping*
Mosaics: main steps

- Collect correspondences (manually)
- Solve for homography matrix $H$
  - Least squares solution
- **Warp content from one image frame to the other to combine: say im1 into im2 reference frame**
  - Determine bounds of the new combined image
    - Where will the corners of im1 fall in im2’s coordinate frame?
    - We will attempt to lookup colors for any of these positions we can get from im1.
  - Compute coordinates in im1’s reference frame (via homography) for all points in that range: $H^{-1}$
  - Lookup all colors for all these positions from im1
    - Inverse warp: interp2 (watch for nans: isnan)
- Overlay im2 content onto the warped im1 content.
  - Careful about new bounds of the output image: minx, miny
Mosaics: main steps

• Collect correspondences (manually)
• Solve for homography matrix H
  – Least squares solution
• Warp content from one image frame to the other to combine: say im1 into im2 reference frame
  – Determine bounds of the new combined image
    • Where will the corners of im1 fall in im2’s coordinate frame?
    • We will attempt to lookup colors for any of these positions we can get from im1
      • meshgrid
  – Compute coordinates in im1’s reference frame (via homography) for all points in that range: $H^{-1}$
  – Lookup all colors for all these positions from im1
    • Inverse warp: interp2 (watch for nans: isnan)
• Overlay im2 content onto the warped im1 content.
  – Careful about new bounds of the output image: minx, miny
Mosaics: main steps

• Collect correspondences (manually)
• Solve for homography matrix H
  – Least squares solution
• Warp content from one image frame to the other to combine: say im1 into im2 reference frame
  – Determine bounds of the new combined image
    • Where will the corners of im1 fall in im2’s coordinate frame?
    • We will attempt to lookup colors for any of these positions we can get from im1. : \texttt{meshgrid}
      – Compute coordinates in im1’s reference frame (via homography) for all points in that range: \( H^{-1} \)
      – Lookup all colors for all these positions from im1
        • Inverse warp: \texttt{interp2 (watch for nans : \texttt{isnan})}
• Overlay im2 content onto the warped im1 content.
  – Careful about new bounds of the output image: minx, miny
Use interp2 to ask for the colors (possibly interpolated) from im1 at all the positions needed in im2’s reference frame.
Mosaics: main steps

• Collect correspondences (manually)
• Solve for homography matrix $H$
  – Least squares solution
• Warp content from one image frame to the other to combine: say $im_1$ into $im_2$ reference frame
  – Determine bounds of the new combined image
    • Where will the corners of $im_1$ fall in $im_2$’s coordinate frame?
    • We will attempt to lookup colors for any of these positions we can get from $im_1$. :meshgrid
  – Compute coordinates in $im_1$’s reference frame (via homography) for all points in that range: $H^{-1}$
  – Lookup all colors for all these positions from $im_1$
    • Inverse warp: interp2 (watch for nans : isnan)
• Overlay $im_2$ content onto the warped $im_1$ content.
  – Careful about new bounds of the output image: minx, miny
Fun with homographies

Original image

St. Petersburg
photo by A. Tikhonov

Virtual camera rotations
Analyzing patterns and shapes

What is the shape of the b/w floor pattern?

The floor (enlarged)

Automatically rectified floor

Slide from Criminisi
Analyzing patterns and shapes

From Martin Kemp *The Science of Art* (manual reconstruction)

2 patterns have been discovered!

Slide from Criminisi
Analyzing patterns and shapes

What is the (complicated) shape of the floor pattern?

*St. Lucy Altarpiece, D. Veneziano*

Slide from Criminisi
Analyzing patterns and shapes

From Martin Kemp, *The Science of Art* (manual reconstruction)

Slide from Criminisi
Analyzing patterns and shapes

The Ambassadors by Hans Holbein the Younger, 1533
Julian Beever: Manual Homographies

http://users.skynet.be/J.Beever/pave.htm
changing camera center

Does it still work?

synthetic PP

PP1

PP2
Planar scene (or far away)

PP3 is a projection plane of both centers of projection, so we are OK!

This is how big aerial photographs are made
Planar mosaic
RANSAC and projections

© Jeffrey Martin (jeffrey-martin.com)

most slides from Steve Seitz, Rick Szeliski, and Alexei A. Efros; James Hays
Suppose we have estimated correspondences, but some are bad

What do we do about the “bad” matches?
RANdom SAmple Consensus

Select one match, count inliers
RANdom SAmpIe Consensus

Select one match, count inliers
Least squares fit

Find “average” translation vector
RANSAC for estimating homography

RANSAC loop:
1. Select four feature pairs (at random)
2. Compute homography $H$ (exact)
3. Compute inliers where $SSD(p_i', H p_i) < \varepsilon$
4. Keep largest set of inliers
5. Re-compute least-squares $H$ estimate on all of the inliers
The key idea is *not* that there are more inliers than outliers, but that the outliers are wrong in *different* ways.
RANSAC
Do we have to project onto a plane?
Full Panoramas

What if you want a $360^\circ$ field of view?

mosaic Projection Cylinder
Cylindrical projection

- Map 3D point \((X, Y, Z)\) onto cylinder
  \[
  (\tilde{x}, \tilde{y}, \tilde{z}) = \frac{1}{\sqrt{X^2+Z^2}}(X, Y, Z)
  \]

- Convert to cylindrical coordinates
  \[
  (\sin\theta, h, \cos\theta) = (\tilde{x}, \tilde{y}, \tilde{z})
  \]

- Convert to cylindrical image coordinates
  \[
  (\tilde{x}, \tilde{y}) = (f\theta, fh) + (\tilde{x}_c, \tilde{y}_c)
  \]
Cylindrical Projection
Inverse Cylindrical projection

\[\theta = \frac{(x_{cyl} - x_c)}{f}\]
\[h = \frac{(y_{cyl} - y_c)}{f}\]
\[\hat{x} = \sin \theta\]
\[\hat{y} = h\]
\[\hat{z} = \cos \theta\]
\[x = f\hat{x}/\hat{z} + x_c\]
\[y = f\hat{y}/\hat{z} + y_c\]
Cylindrical panoramas

Steps
- Reproject each image onto a cylinder
- Blend
- Output the resulting mosaic
What if you don’t know the camera rotation?

• Solve for the camera rotations
  – Note that a rotation of the camera is a translation of the cylinder!
Assembling the panorama

Stitch pairs together, blend, then crop
Problem: Drift

Vertical Error accumulation
- small (vertical) errors accumulate over time
- apply correction so that sum = 0 (for 360° pan.)

Horizontal Error accumulation
- can reuse first/last image to find the right panorama radius
Full-view (360°) panoramas
Other projections are possible

You can stitch on the plane and then warp the resulting panorama
  • What’s the limitation here?
Or, you can use these as stitching surfaces
  • But there is a catch…
Cylindrical reprojection

Image 384x300  \( f = 180 \) (pixels)  \( f = 280 \)  \( f = 380 \)

Focal length – the dirty secret
What's your focal length, buddy?

Focal length is (highly!) camera dependant

- Can get a rough estimate by measuring FOV:

![Diagram showing the relationship between focal length (f), field of view (θ/2), and image width (W/2).]

- Can use the EXIF data tag (might not give the right thing)
- Can use several images together and try to find f that would make them match
- Can use a known 3D object and its projection to solve for f
- Etc.

There are other camera parameters too:

- Optical center, non-square pixels, lens distortion, etc.
Spherical projection

- Map 3D point \((X,Y,Z)\) onto sphere

\[
(\hat{x}, \hat{y}, \hat{z}) = \frac{1}{\sqrt{X^2 + Y^2 + Z^2}}(X,Y,Z)
\]

- Convert to spherical coordinates

\[
(\sin \theta \cos \phi, \sin \phi, \cos \theta \cos \phi) = (\hat{x}, \hat{y}, \hat{z})
\]

- Convert to spherical image coordinates

\[
(\tilde{x}, \tilde{y}) = (f \theta, f h) + (\tilde{x}_c, \tilde{y}_c)
\]
Spherical Projection
Inverse Spherical projection

\[ \theta = \frac{(x_{sp} - x_c)}{f} \]
\[ \varphi = \frac{(y_{sp} - y_c)}{f} \]
\[ \hat{x} = \sin \theta \cos \varphi \]
\[ \hat{y} = \sin \varphi \]
\[ \hat{z} = \cos \theta \cos \varphi \]
\[ x = \frac{f \hat{x}}{\hat{z}} + x_c \]
\[ y = \frac{f \hat{y}}{\hat{z}} + y_c \]
3D rotation

Rotate image before placing on unrolled sphere

\[
\begin{align*}
\theta &= \frac{(x_{\text{sph}} - x_c)}{f} \\
\varphi &= \frac{(y_{\text{sph}} - y_c)}{f} \\
\hat{x} &= \sin \theta \cos \varphi \\
\hat{y} &= \sin \varphi \\
\hat{z} &= \cos \theta \cos \varphi \\
x &= \frac{f \hat{x}}{\hat{z}} + x_c \\
y &= \frac{f \hat{y}}{\hat{z}} + y_c
\end{align*}
\]

\[p = Rp\]
Full-view Panorama
Distortion

Radial distortion of the image
- Caused by imperfect lenses
- Deviations are most noticeable for rays that pass through the edge of the lens
Radial distortion

Correct for “bending” in calibrated wide field of view lenses

\[ \hat{r}^2 = \hat{x}^2 + \hat{y}^2 \]
\[ \hat{x}' = \hat{x} / (1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4) \]
\[ \hat{y}' = \hat{y} / (1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4) \]
\[ x = f \hat{x}' / \hat{z} + x_c \]
\[ y = f \hat{y}' / \hat{z} + y_c \]

Use this instead of normal projection if we know radial distortion coefficients \( \kappa_1 \) and \( \kappa_2 \)
Polar Projection

Extreme “bending” in ultra-wide fields of view

\[ \hat{r}^2 = \hat{x}^2 + \hat{y}^2 \]

\[(\cos \theta \sin \phi, \sin \theta \sin \phi, \cos \phi) = s (x, y, z)\]

Equations become

\[
x' = s \phi \cos \theta = s \frac{x}{r} \tan^{-1} \frac{r}{z},
\]

\[
y' = s \phi \sin \theta = s \frac{y}{r} \tan^{-1} \frac{r}{z},
\]
Optimize over more parameters: Bundle Adjustment

New images initialised with rotation, focal length of best matching image
Optimize over more parameters: Bundle Adjustment

New images initialised with rotation, focal length of best matching image
Multi-band Blending

Burt & Adelson 1983

- Blend frequency bands over range $\propto \lambda$
Project 5: Bells and Whistles

Blending and Compositing

• use homographies to combine images or video and images together in an interesting (fun) way. E.g.
  – put fake graffiti on buildings or chalk drawings on the ground
  – replace a road sign with your own poster
  – project a movie onto a building wall
  – etc.
Capture creative/cool/bizzare panoramas

- Example from UW (by Brett Allen):

- Ever wondered what is happening inside your fridge while you are not looking?
Bells and Whistles

Video Panorama

- Capture two (or more) stationary videos (either from the same point, or of a planar/far-away scene). Compute homography and produce a video mosaic. Need to worry about synchronization (not too hard).
- e.g. capturing a football game from the sides of the stadium
More panorama / homography examples

Ben Hollis, 2004

Ben Hollis, 2004

Matt Pucevich, 2004

Eunjeong Ryu (E.J), 2004