Image Stitching 2
Project 1

- Appreciate using the link without attachment
- Check the permission (use incognito to click the link)
Project 2

Creating your own panorama!

1. You can do by yourself or form a group (at most 3).

2. We have 2 pairs of “a camera and a tripod” to check out if you need them.
   (likely use http://youcanbook.me)

3. Start early otherwise, you won’t have time to take your images…
Motion models

Translation: 2 unknowns
Affine: 6 unknowns
Perspective (Homography): 8 unknowns
3D rotation: 3 unknowns
3D Rotation

(u₀, v₀)
3D Rotation

- Compute \((u_1, v_1)\) given \((u_0, v_0)\)
3D Rotation

- Compute \((u_1, v_1)\) given \((u_0, v_0)\)
3D Rotation
3D rotation to make panorama

The mosaic has a natural interpretation in 3D
  • The images are reprojected onto a common plane
  • The mosaic is formed on this plane
3D rotation to make panorama

• Compute \((u_1, v_1)\) given \((u_0, v_0)\)
3D Rotation \((u_0, v_0) \rightarrow (u_1, v_1)\)

Projection equations
1. Project from screen to 3D ray
2. Rotate the ray by camera motion
3. Project back into new (source) image
3D Rotation \((u_0, v_0) \rightarrow (u_1, v_1)\)

Projection equations
1. Project from screen to 3D ray
   \[(x_0, y_0, z_0) = (u_0-u_c, v_0-v_c, f)\]
2. Rotate the ray by camera motion
3. Project back into new (source) image
3D Rotation \((u_0, v_0) \rightarrow (u_1, v_1)\)

Projection equations
1. Project from screen to 3D ray
   \[(x_0, y_0, z_0) = (u_0-u_c, v_0-v_c, f)\]
2. Rotate the ray by camera motion
   \[(x_1, y_1, z_1) = R_{01} (x_0, y_0, z_0)\]
3. Project back into new (source) image
3D Rotation \((u_0, v_0) \rightarrow (u_1, v_1)\)

Projection equations
1. Project from screen to 3D ray
\[(x_0, y_0, z_0) = (u_0-u_c, v_0-v_c, f)\]
2. Rotate the ray by camera motion
\[(x_1, y_1, z_1) = R_{01} (x_0, y_0, z_0)\]
3. Project back into new (source) image
\[(u_1, v_1) = (f'x_1/z_1+u'_c, f'y_1/z_1+v'_c)\]
Application for Video Coding

Convert masked images into a background sprite for content-based coding
Why cannot we use this technique to create a panorama?
Motion between planar screen and non-planar screen

What if you want a 360° panorama?

Projection Cylinder
Non-planar reprojection screen...

360-degrees Panorama

+         +         +...+
Cylindrical panoramas
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
  
  \[
  (\tilde{x}, \tilde{y}) = (s\theta, sh) + (\bar{x}_c, \bar{y}_c)
  \]

  \(-s\) defines size of the final image
Cylindrical Warping

Projection Cylinder

\((x_{cyl}, y_{cyl})\)
Cylindrical warping

Given focal length $f$ and image center $(x_c, y_c)$

\[
\theta = \frac{(x_{cyl} - x_c)}{f} \\
h = \frac{(y_{cyl} - y_c)}{f} \\
\hat{x} = \sin \theta \\
\hat{y} = h \\
\hat{z} = \cos \theta \\
x = f \hat{x}/\hat{z} + x_c \\
y = f \hat{y}/\hat{z} + y_c
\]
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
  \[
  (\tilde{x}, \tilde{y}) = (s\theta, s\phi) + (\tilde{x}_c, \tilde{y}_c)
  \]
  - \(s\) defines size of the final image
  - often convenient to set \(s = \text{camera focal length}\)
Spherical Warping
Spherical Warping

Rotate image before placing on unrolled sphere

\[
\begin{align*}
\theta &= \frac{(x_{sph} - x_c)}{f} \\
\phi &= \frac{(y_{sph} - y_c)}{f} \\
\hat{x} &= \sin \theta \cos \phi \\
\hat{y} &= \sin \phi \\
\hat{z} &= \cos \theta \cos \phi \\
x &= \frac{f \hat{x}}{\hat{z}} + x_c \\
y &= \frac{f \hat{y}}{\hat{z}} + y_c
\end{align*}
\]
Cylinder vs Sphere...
Motions

Planar Plane of Projection
- Translation
- Affine
- Homography
- 3D Rotation

Cylindrical Plane of Projection
- 3D Rotation

Spherical Plane of Projection
- 3D Rotation
Motions

Planar Plane of Projection
- Translation
- Affine
- Homography
- 3D Rotation

Cylindrical Plane of Projection
- 3D Rotation

Spherical Plane of Projection
- 3D Rotation
# 3D Rotation Cases

<table>
<thead>
<tr>
<th>Projection screen</th>
<th>Plane</th>
<th>Cylinder</th>
<th>Sphere</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformation</td>
<td>Linear</td>
<td>Non-linear</td>
<td>Non-linear</td>
</tr>
<tr>
<td>Application</td>
<td>Wide angle Video-coding</td>
<td>Panorama ...</td>
<td>Panorama ...</td>
</tr>
</tbody>
</table>
Panorama Stitcher w/ 3D Rotational Motions

Music: Kevin MacLeod
For Project 2

Assume horizontal cameras
Pre-warp images onto spherical images

(unit sphere)

Spherical image
For Project 2

Assume horizontal cameras
Pre-warp images onto spherical images
For Project 2

Assume horizontal cameras
Pre-warp images onto spherical images
For Project 2

Assume horizontal cameras
Pre-warp images onto spherical images
For Project 2

1D translation problem on warped images
For Project 2

1D translation problem on warped images
But, solve for 2D translation to allow errors
Today’s lecture

Image alignment and stitching
• motion models
• image warping
• point-based alignment
• complete mosaics (global alignment)
• compositing and blending
• ghost and parallax removal
Computing transformations

• Given a set of matches between images A and B
  – How to compute the transform T from A to B?
  – Find transform T that best “explains” the matches
If you know which motion...

- Translation: 2 unknowns
- Affine: 6 unknowns
- Perspective (Homography): 8 unknowns
- 3D rotation: 3 unknowns
Simple case: translations

How do we solve for $(x_t, y_t)$?
Simple case: translations

Displacement of match $i = (x'_i - x_i, y'_i - y_i)$

Simple strategy to compute $(x_t, y_t)$?
Just take the average!
Verified by the least squares solution...
Simple case: translations

\[ x_i + x_t = x'_i \]
\[ y_i + y_t = y'_i \]
Least squares formulation

• Can also write as a matrix equation

\[
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
1 & 0 \\
0 & 1 \\
\vdots \\
1 & 0 \\
0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
x_t \\
y_t \\
\end{bmatrix}
= 
\begin{bmatrix}
x'_1 - x_1 \\
y'_1 - y_1 \\
x'_2 - x_2 \\
y'_2 - y_2 \\
\vdots \\
x'_n - x_n \\
y'_n - y_n \\
\end{bmatrix}
\]
Least squares

\[ A^t = b \]
Least squares

\[ At = b \]

• Find \( t \) that minimizes

\[ \| At - b \|^2 \]
Least squares

\[ At = b \]

• Find \( t \) that minimizes

\[ \| At - b \|^2 \]

• To solve, form the *normal equations*

\[ A^T At = A^T b \]

\[ t = (A^T A)^{-1} A^T b \]
Affine transformations

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

How many unknowns?
How many equations per match?
How many matches do we need?
Affine transformations

Matrix form

\[
\begin{bmatrix}
  x_1 & y_1 & 1 & 0 & 0 & 0 \\
  0 & 0 & 0 & x_1 & y_1 & 1 \\
  x_2 & y_2 & 1 & 0 & 0 & 0 \\
  0 & 0 & 0 & x_2 & y_2 & 1 \\
  \vdots \\
  x_n & y_n & 1 & 0 & 0 & 0 \\
  0 & 0 & 0 & x_n & y_n & 1 \\
\end{bmatrix}
\begin{bmatrix}
  a \\
  b \\
  c \\
  d \\
  e \\
  f \\
\end{bmatrix}
= 
\begin{bmatrix}
  x'_1 \\
  y'_1 \\
  x'_2 \\
  y'_2 \\
  \vdots \\
  x'_n \\
  y'_n \\
\end{bmatrix}
\]
Least squares

\[ \mathbf{A} \mathbf{t} = \mathbf{b} \]

- Find \( \mathbf{t} \) that minimizes

\[ \| \mathbf{A} \mathbf{t} - \mathbf{b} \|^2 \]

- To solve, form the *normal equations*

\[
\mathbf{A}^T \mathbf{A} \mathbf{t} = \mathbf{A}^T \mathbf{b}
\]

\[
\mathbf{t} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b}
\]
Homographies

To unwarp (rectify) an image

- solve for homography $H$ given $p$ and $p'$
- solve equations of the form: $wp' = Hp$
  - linear in unknowns: $w$ and coefficients of $H$
  - $H$ is defined up to an arbitrary scale factor
  - how many points are necessary to solve for $H$?
Solving for homographies

\[
\begin{bmatrix}
x'_i \\
y'_i \\
1
\end{bmatrix} \Rightarrow \begin{bmatrix}
h_{00} & h_{01} & h_{02} \\
h_{10} & h_{11} & h_{12} \\
h_{20} & h_{21} & h_{22}
\end{bmatrix} \begin{bmatrix}
x_i \\
y_i \\
1
\end{bmatrix}
\]

\[
x'_i = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}}
\]

\[
y'_i = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}}
\]

Not linear!

\[
x'_i(h_{20}x_i + h_{21}y_i + h_{22}) = h_{00}x_i + h_{01}y_i + h_{02}
\]

\[
y'_i(h_{20}x_i + h_{21}y_i + h_{22}) = h_{10}x_i + h_{11}y_i + h_{12}
\]
Solving for homographies

\[
x'_i(h_{20}x_i + h_{21}y_i + h_{22}) = h_{00}x_i + h_{01}y_i + h_{02}
\]
\[
y'_i(h_{20}x_i + h_{21}y_i + h_{22}) = h_{10}x_i + h_{11}y_i + h_{12}
\]
Solving for homographies

Define a least squares problem:

\[
\begin{bmatrix}
x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\
0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\
x_n & y_n & 1 & 0 & 0 & 0 & -x'_n x_n & -x'_n y_n & -x'_n \\
0 & 0 & 0 & x_n & y_n & 1 & -y'_n x_n & -y'_n y_n & -y'_n \\
\end{bmatrix}
\begin{bmatrix}
h_{00} \\
h_{01} \\
h_{02} \\
h_{10} \\
h_{11} \\
h_{12} \\
h_{20} \\
h_{21} \\
h_{22} \\
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\]

\[
A = \begin{bmatrix}
x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\
0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\
x_n & y_n & 1 & 0 & 0 & 0 & -x'_n x_n & -x'_n y_n & -x'_n \\
0 & 0 & 0 & x_n & y_n & 1 & -y'_n x_n & -y'_n y_n & -y'_n \\
\end{bmatrix}
\]

\[
h = \begin{bmatrix}
h_{00} \\
h_{01} \\
h_{02} \\
h_{10} \\
h_{11} \\
h_{12} \\
h_{20} \\
h_{21} \\
h_{22} \\
\end{bmatrix}
\]

Define a least squares problem:

\[
\text{minimize } \| Ah - 0 \|^2
\]

- Since \( h \) is only defined up to scale, solve for unit vector \( \hat{h} \)
- Solution: \( \hat{h} = \) eigenvector of \( A^T A \) with smallest eigenvalue
- Works with 4 or more points
Singular Value Decomposition

\[ M = U \Sigma V^* \]

Consider the 4×5 matrix

\[
M = \begin{bmatrix}
1 & 0 & 0 & 0 & 2 \\
0 & 0 & 3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 & 0
\end{bmatrix}
\]

A singular value decomposition of this matrix is given by \( U \Sigma V^* \)

\[
U = \begin{bmatrix}
0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 \\
1 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
\Sigma = \begin{bmatrix}
4 & 0 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 & 0 \\
0 & 0 & \sqrt{5} & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
V^* = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
\sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\
0 & 0 & 0 & 1 & 0 \\
-\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2}
\end{bmatrix}
\]
Singular Value Decomposition

\[ M = U \Sigma V^* \]

Consider the 4×5 matrix

\[
M = \begin{bmatrix}
1 & 0 & 0 & 0 & 2 \\
0 & 0 & 3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 & 0
\end{bmatrix}
\]

A singular value decomposition of this matrix is given by \( U \Sigma V^* \)

\[
U = \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & -1 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
\Sigma = \begin{bmatrix}
4 & 0 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 & 0 \\
0 & 0 & \sqrt{5} & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
V^* = \begin{bmatrix}
\sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\
0 & 0 & 1 & 0 & 0 \\
-\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2}
\end{bmatrix}
\]

\[
MM^* = (U \Sigma V^*) (V \Sigma^* U^*)
= U (\Sigma \Sigma^*) U^*
= U (\Sigma') U^*
\]

\[
(MM^*) = U \Sigma' U^{-1}
\]
Singular Value Decomposition

\[ M = U \Sigma V^* \]

Consider the 4x5 matrix

\[
M = \begin{bmatrix}
1 & 0 & 0 & 0 & 2 \\
0 & 0 & 3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 & 0 \\
\end{bmatrix}
\]

A singular value decomposition of this matrix is given by \( U \Sigma V^* \)

\[
U = \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & -1 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
\Sigma = \begin{bmatrix}
4 & 0 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 & 0 \\
0 & 0 & \sqrt{5} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
V^* = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
\sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\
0 & 0 & 0 & 1 & 0 \\
-\sqrt{0.8} & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
MM^* = (U \Sigma V^*)(V \Sigma^* U^*) = U(\Sigma \Sigma^*) U^* = U(\Sigma') U^*
\]

\[
(MM^*) = U \Sigma' U^{-1}
\]

\[
M^* M = (V \Sigma^* U^*)(U \Sigma V^*) = V(\Sigma^* \Sigma) V^* = V(\Sigma'' ) V^*
\]

\[
(M^* M) = V \Sigma'' V^{-1}
\]

[ Wikipedia ]