Image Formation

January 8, 2024

1 Camera Model

- Draw pinhole camera model
  - Note that all rays go through the optical center
  - Equivalent for thin lens is the focal plane
    - Point at which parallel rays converge independent of angle with optical axis more info
  - 1/xo + 1/xi = 1/f
- 1D Camera
  - Maps YZ to y
  - Simple projection
  - We can also think of this as a projective space
  - Maps rays onto a plane
  - Note that we can't see points that are orthogonal to the optical (z) axis that are on the so called prinicipal plane
  - Note also that multiple points are mapped to the same point on the image
    - We can't determine the scale of the object from the image alone

1.1 Big Ideas

- Taken from Multiview Geometry
- When we look at a picture, squares are not squares and circles are not circles
- This is because our eyes and cameras map planar objects through Projective Transformations
- Specifically we map onto a projective plane $\mathbb{P}^2$

1.1.1 Projective Plane

- Think of a set of rays in $\mathbb{R}^3$
- All rays go through a single point
  - Sometimes called
    - Camera center
    - Focal point
  - We typically treat this as the origin
- The set of vectors/rays $k(x_1, x_2, x_3)^T$ are indistinguishable
- Consider where the rays cross when they hit a plane at, say, $x_3 = 1$
  - This is called the projective plane $\mathbb{P}^2$

- These rays represent a single point in $\mathbb{P}^2$
- Two non-identical rays lie on exactly one plane
  - Similar to idea that two distinct points uniquely define a line
- Any two planes intersect one ray
  - Two lines always intersect in a point
- Points and lines can be obtained by intersecting the rays and planes by $x_3 = 1$
- Ideal points and the plane representing l_∞ are parallel to the imaging plane $x_3 = 1$
- Cameras can also be though of as a central projection (along $x_3 = 1$)
- Maps $\mathbb{P}^3$ to $\mathbb{P}^2$
- From $(X, Y, Z, T)^T$ to

$P = [I_{3\times3}|0_3]$

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = P_{3\times4} \begin{bmatrix} X \\ Y \\ Z \\ T \end{bmatrix}$$

1.1.2 Projective Transformation Invariants

- What properties are preserved under projective transformation?
  - Not shape
    - Circles may appear as ellipses
  - Not lengths - radii of circle stretched by different amounts
  - Not angles, distance, ratios of distances - none are preserved
  - Straightness is preserved
    - Straight lines remain straight lines
  - We could define a projective transformation of a plane as any mapping of points that preserves straight lines

1.2 Homogenous Representations

- We'll work with 2D images
- We can specify a location on the image as an (x, y) pair
- This could represent anywhere in a Euclidian 2D space

1.2.1 Lines

- A line in a plane can be represented as $ax + by + c = 0$
- The choice of a, b, and c give rise to different lines
- We could represent the line as a vector (a, b, c)
- Something is wrong: we know lines only have 2 degrees of freedom
- However, note that $(ka)x + (kb)y + (kc) = 0$ represents the same line for any $k \neq 0$
- We can consider the vectors (a, b, c) and (ka, kb, kc) to be equivalent
- Any particular vector (a, b, c) is merely a representative of a class of vectors that are equivalent

- The equivalence class in $\mathbb{R}^3 - (0,0,0)^T$ forms a projective space $\mathbb{P}^2$
  - The notation $-(0,0,0)^T$ merely means that the vector $(0,0,0)$ is excluded

1.3 Homogenous Representation of Points

- A point $u = (x,y)^T$ lines on the line $l = (a,b,c)^T$ only if $ax + by + c = 0$
- We can represent this as an inner product $(x,y,1)(a,b,c)^T = (x,y,1)l = 0$
- Note that in this case we represented a two dimensional point by appending a 1 $(x,y,1)$ such that the final coordinate was 1
- This is sometimes called an *augmented vector*
- Note that for any non-zero k $(kx,ky,k)l = 0$ if and only if $(x,y,1)l = 0$
- Therefore all of the vectors (kx,ky,k) are a representation of the point $(x,y)^T$ in $\mathbb{R}^2$
- Thus any point of the form $u = (x_1,x_2,x_3)^T$ represents the point $(x_1/x_3, x_2/x_3)$ in $\mathbb{R}^2$
- Generally these 3D vector representations of 2D points in the projective space are called *homogenous vectors*
- If they are scaled so that the last element is 1, the resulting vector is called an *inhomogenous vector*
  - Sometimes this term is used only for the 2D vector (x,y)
- Much like we sometimes use a complex number with 2 degrees of freedom to represent a point on the unit circle, there are often advantages to using an extra and seemingly uneeded degree of freedom
- It can simplify the math and computation
- For example affine operations become inner products

1.4 Ideal Points

- Note that we have a problem if the last element is 0
- These are called *ideal points* or *points at infinity*
- These represent points that are on the principal plane
- Even though they are "at infinity" we can represent them with homogeneous coordinates
- Thus homogenous coordinates are more general than a Euclidean space

1.5 Homogeneity

1.5.1 Euclidean Geometry

- In Euclidean geometry all points are the same
- There is no special point
- The whole space is homogenous
- The origin is arbitrary
- We could choose any point and orientation as the origin
- We can easily transform from one origin to another with a change of coordinates
- Called a **Euclidean Transform**
- If we translate, rotate, and *stretch* is called an Affine transform
- Points at infinity remain at infinity

1.5.2 Projective Geometry

- Points at infinity are no different

- We can easily represent them
- Projective space is uniform
- A **Projective Transformation** can be represented as a non-singular matrix multiplication
- Can move points at infinity to other points
- Points at infinity are not preserved

1.6 Cameras as Points

- Points in $\mathbb{P}^3$ are mapped to $\mathbb{P}^2$
- All points in a ray pass through the center of projection
- All points along the ray are essentially equal
- You can think of the ray through the projection center as representing the image point
- All image points are the same as set of rays going through the camera center
- If we repreent the ray from $(0,0,0,1)^T$ through $(X,Y,Z,T)^T$ by first three coordinates
- The only important thing is the camera center
- This alone determines the set of rays that form the image
- Two images from the same point in space are projectively equivalent
  - They can be mapped from on to the other by a projective transformation
  - Doesn't require any information about the 3D points
  - Don't need to know the camera center
- See Figure 1.1 on Page 8 of Multiview Geometry

[ ]:
```
import cv2 as cv
import os
import matplotlib.pyplot as plt
import numpy as np
import os
import PSUCV as pcv
#print(os.getpwd())

#imagePath = os.path.join(os.path.expanduser('~'),'Temp','Test.jpg')
#imageBGR = cv2.imread(imagePath)
#imageRGB = cv2.cvtColor(imageBGR, cv2.COLOR_BGR2RGB)

imageBGR  = cv.imread('Bubble.jpeg')
imageRGB = cv.cvtColor(imageBGR, cv.COLOR_BGR2RGB)
#imageGray = cv.cvtColor(imageBGR, cv.COLOR_BGR2GRAY)
```

[ ]:
```
figure = plt.figure()
figure.clf()
figure.set_size_inches(15,10)
axes = figure.add_subplot(121)
axes.imshow(imageBGR)
axes.set_title('OpenCV BGR')

axes = figure.add_subplot(122)
axes.imshow(imageRGB)
axes.set_title('RGB')
```

```
plt.show()

image = imageRGB
```

[ ]:

1.7 2D Transformations (2.1.2)

1.7.1 Translation

$x' = x + t$

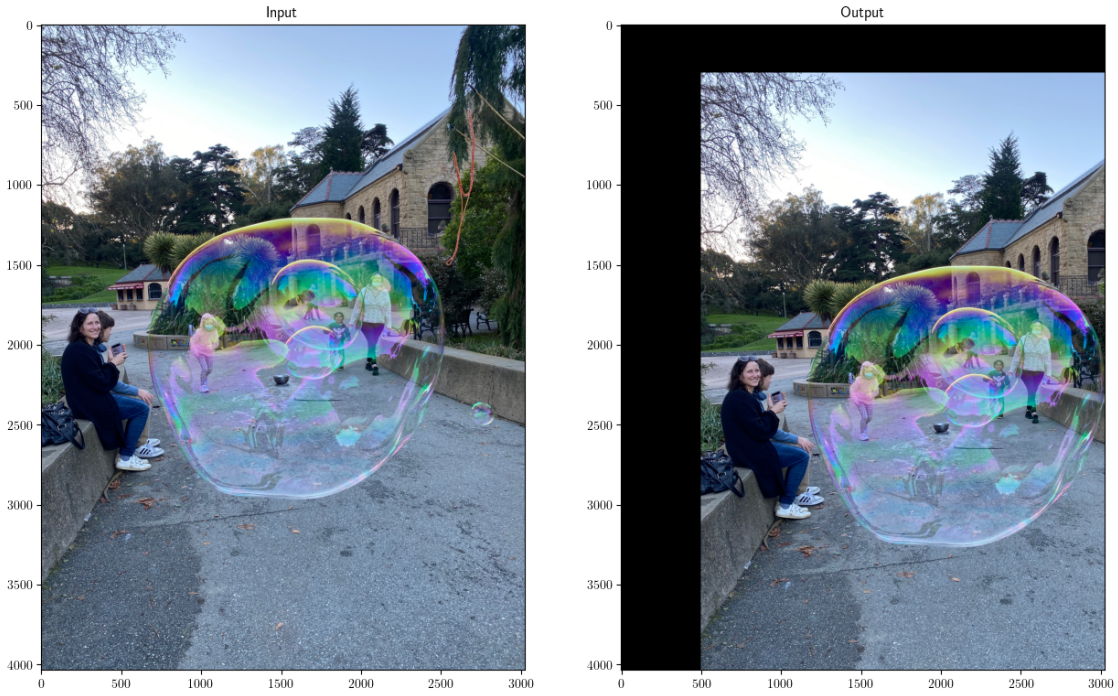
$\bar{x}' = \begin{bmatrix} I & t \\ 0^T & 1 \end{bmatrix} \bar{x}$

- Another handy feature of homogenous coordinates
- Affine transformations (linear transformation and translation) combined into a single matrix multiplication
- Note that by convention the origin is at (0,0) in the image
  - Top left corner
  - Vertical axies is inverted (by convention)

[ ]:
```
M = np.eye(3)
M[0,2] = 500 # Pixels along vertical (x) axis
M[1,2] = 300 # Pixels along vertical (y) axis
```

```
print(M)

imageWarped = cv.warpPerspective(image,M,image.shape[:2][::-1])

pcv.DisplaySideBySide(image,imageWarped)
```

```
[[  1.   0. 500.]
 [  0.   1. 300.]
 [  0.   0.   1.]]
```



1.7.2 Rotation + Translation

$x' = Rx + t = \begin{bmatrix} R & t \end{bmatrix} \bar{x}$

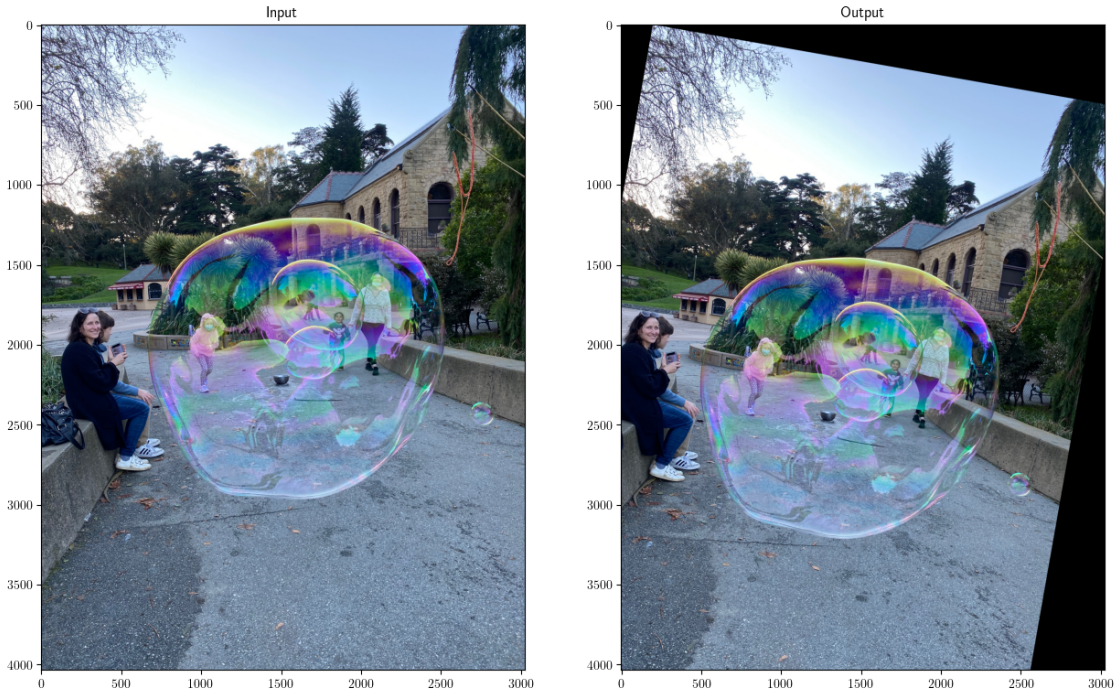
$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$

- Again, usually translation isn't actually done
- R is orthonormal
  - $RR^T = I$
  - $|R| = 1$

[ ]:
```
theta = 10.0*(np.pi/180.0)

M = np.array([[np.cos(theta),-np.sin(theta),200],[np.sin(theta),np.cos(theta),0.
↪0],[0.0,0.0,1.0]])
```

```
R = M[:2,:2]
print(M)

imageWarped = cv.warpPerspective(image,M,image.shape[:2][::-1])

pcv.DisplaySideBySide(image,imageWarped)
```

```
[[ 9.84807753e-01 -1.73648178e-01  2.00000000e+02]
 [ 1.73648178e-01  9.84807753e-01  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```



1.7.3 Scaled Rotation + Translation

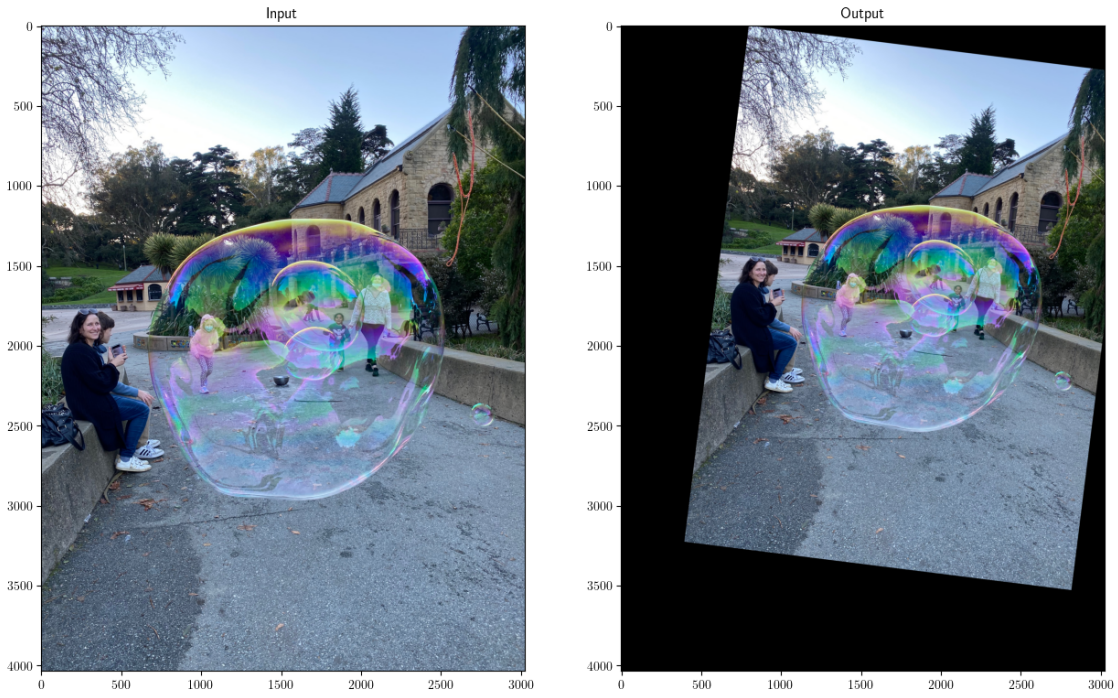
$$x' = sRx + t = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \bar{x}$$

- Preserves angles between lines

[ ]:
```
a = 0.8
b = 0.1
tx = 800
ty = 0

M = np.array([[a,-b,tx],[b,a,ty],[0.0,0.0,1.0]])
print(M)
```

```
imageWarped = cv.warpPerspective(image,M,image.shape[:2][::-1])

pcv.DisplaySideBySide(image,imageWarped)
```

```
[[ 8.e-01 -1.e-01  8.e+02]
 [ 1.e-01  8.e-01  0.e+00]
 [ 0.e+00  0.e+00  1.e+00]]
```



1.7.4 Affine

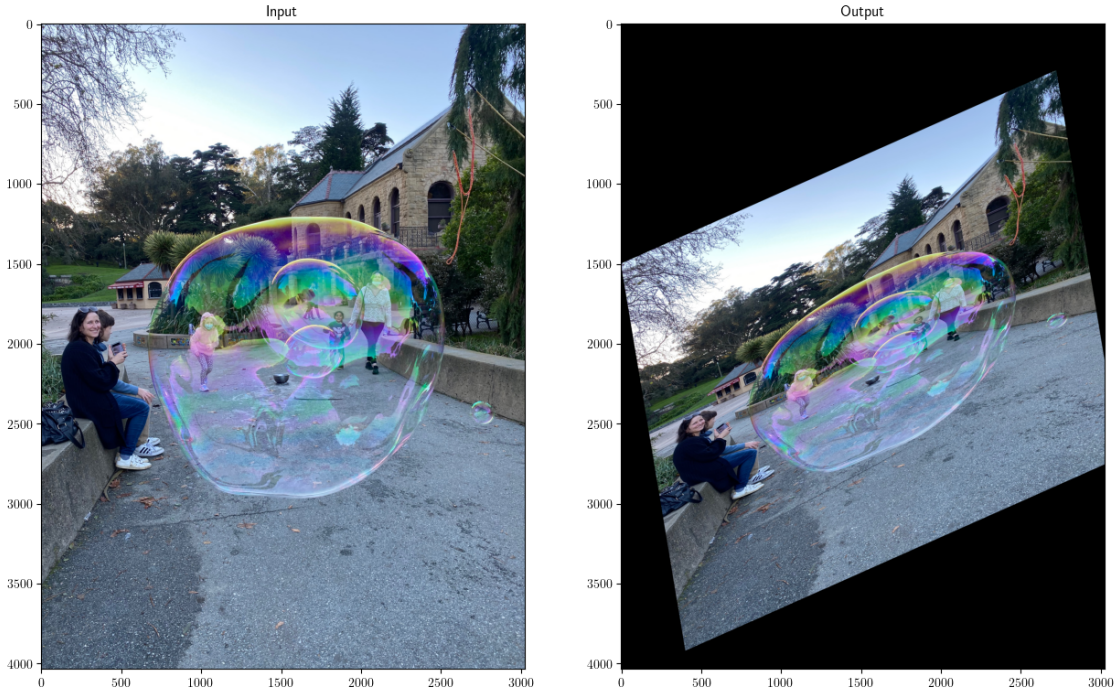
$$x' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \bar{x}$$

- Preserves parallel lines

[ ]:
```
a00 =  0.9
a01 =  0.1
a10 = -0.4
a11 =  0.6
tx = 0
ty = 1500

M = np.array([[a00,a01,tx],[a10,a11,ty],[0.0,0.0,1.0]])
print(M)

imageWarped = cv.warpPerspective(image,M,image.shape[:2][::-1])
```

```
pcv.DisplaySideBySide(image,imageWarped)
```

```
[[ 9.0e-01  1.0e-01  0.0e+00]
 [-4.0e-01  6.0e-01  1.5e+03]
 [ 0.0e+00  0.0e+00  1.0e+00]]
```



1.7.5 Projective

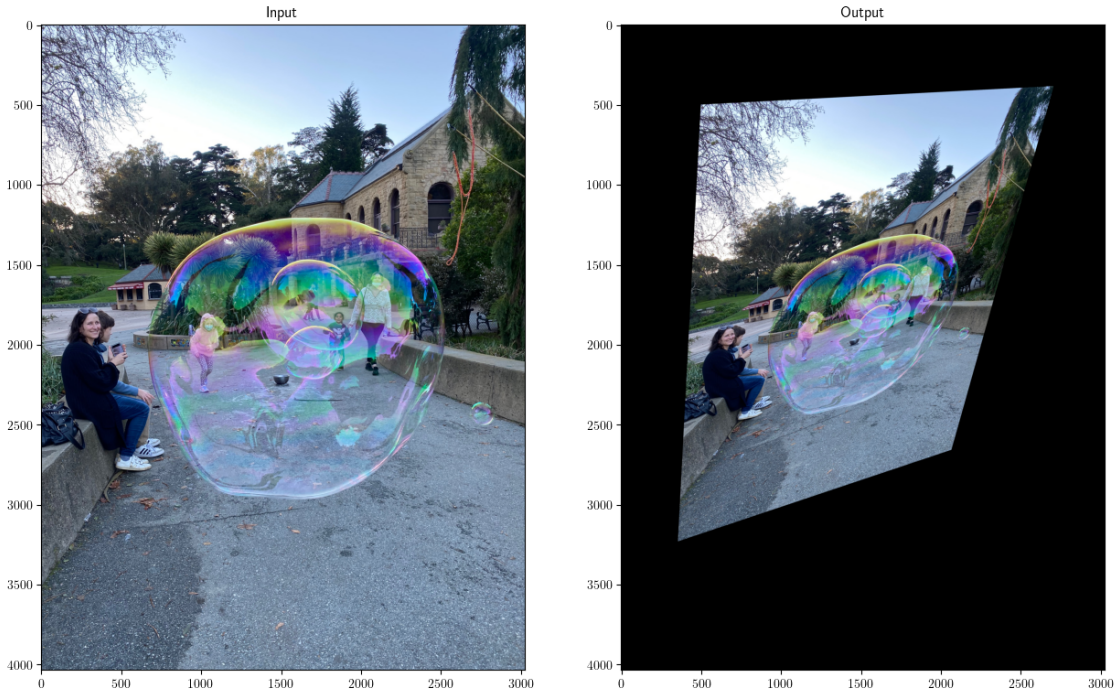
$$\bar{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \bar{x}$$

- Operates on homogenous coordinates
- Arbitrary 3×3 matrix
- Only defined up to scale
- Output vector needs to be normalized to obtain inhomoegenous x
- Preserves straight lines

[ ]:
```
a00 =  1.0
a01 =  0.0
a10 =  0.0
a11 =  1.0
tx  = 500
ty  = 500
a20 = 0.0001
```

```
a21 = 0.0001
a22 = 1.0

M = np.array([[a00,a01,tx],[a10,a11,ty],[a20,a21,a22]])
print(M)

imageWarped = cv.warpPerspective(image,M,image.shape[:2][::-1])

pcv.DisplaySideBySide(image,imageWarped)
```

```
[[1.e+00 0.e+00 5.e+02]
 [0.e+00 1.e+00 5.e+02]
 [1.e-04 1.e-04 1.e+00]]
```



1.8 3D Transformations

We won't need these

[ ]: