4.15 *Lab: Transformations in 2D geometry*

4.15.1 *Our representation for points in the plane*

You are familiar with representing a point (x, y) in the plane by a `{'x','y'}`-vector $\begin{bmatrix} x \\ y \end{bmatrix}$. In this lab, for a reason that will become apparent, we will use an `{'x','y','u'}`-vector $\begin{bmatrix} x \\ y \\ u \end{bmatrix}$. This representation is called *homogeneous coordinates*. We will not be making use of homogeneous coordinates in their full generality; here, the u coordinate will always be 1.

4.15.2 *Transformations*

A geometric transformation will be represented by a matrix M. To apply the transformation to the location of a single point, use matrix-vector multiplication to multiply the matrix by the location vector representing the point.

For example, let's say you want to scale the point by two in the vertical direction. If we were representing points in the plane by 2-vectors $\begin{bmatrix} x \\ y \end{bmatrix}$, we would represent the transformation by the matrix

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

You could apply the transformation to the vector by using matrix-vector multiplication:

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

For example, if the point were located at $(12, 15)$, you would calculate

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 12 \\ 15 \end{bmatrix} = \begin{bmatrix} 12 \\ 30 \end{bmatrix}$$

However, since here we instead represent points in the plane by 3-vectors $\begin{bmatrix} x \\ y \\ u \end{bmatrix}$ (with $u = 1$), you would instead represent the transformation by the matrix

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

You would apply it to a point (x, y) by matrix-vector multiplication:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

For example, to apply it to the point $(12, 15)$, you would calculate

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 12 \\ 15 \\ 1 \end{bmatrix} = \begin{bmatrix} 12 \\ 30 \\ 1 \end{bmatrix}$$

Note that the resulting vector also has a 1 in the u entry.

Suppose we want to apply such a transformation to many points at the same time. As illustrated in Examples 4.11.4 and 4.11.5, according to the matrix-vector definition of matrix-matrix multiplication, to apply the transformation to many points, we put the points together to form a location matrix and left-multiply that location matrix by the matrix representing the transformation:

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{array}{c|c|c|c} 2 & 2 & -2 & -2 \\ 2 & -2 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{array}\right] = \left[\begin{array}{c|c|c|c} 6 & 6 & -6 & -6 \\ 6 & -6 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{array}\right]$$

4.15.3 *Image representation*

In this lab, we will be manipulating images using matrices in Python. In order to do this, we need to represent images as matrices. We represent an image by a set of colored points in the plane.

Colored points

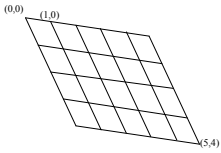
To represent a colored point, we need to specify its location and its color. We will therefore represent a point using two vectors; the *location* vector with labels $\{'x', 'y', 'u'\}$ and the *color* vector with labels $\{'r', 'g', 'b'\}$. The location vector represents the location of the point in the usual way—as an (x, y) pair. The u entry is always 1 for now; later you will see how this is used. For example, the point $(12, 15)$ would be represented by the vector `Vec({'x','y','u'}, {'x':12, 'y':15, 'u':1})`.

The color vector represents the color of the point: the $'r'$, $'g'$, and $'b'$ entries give the intensities for the color channels *red*, *green*, and *blue*. For example, the color red is represented by the function $\{'r' : 1\}$.

Our scheme for representing images

Ordinarily, an image is a regular rectangular grid of rectangular pixels, where each pixel is assigned a color. Because we will be transforming images, we will use a slightly more general representation.

A *generalized image* consists of a grid of generalized pixels, where each generalized pixel is a quadrilateral (not necessarily a rectangle).



The points at the corners of the generalized pixels are identified by pairs (x, y) of integers, which are called *pixel coordinates*. The top-left corner has pixel coordinates (0,0), the corner directly to its right has pixel coordinates (1,0), and so on. For example, the pixel coordinates of the four corners of the top-left generalized pixel are (0,0), (0,1), (1,0), and (1,1).

Each corner is assigned a location in the plane, and each generalized pixel is assigned a color. The mapping of corners to points in the plane is given by a matrix, the *location matrix*. Each corner corresponds to a column of the location matrix, and the label of that column is the pair (x, y) of pixel coordinates of the corner. The column is a `{'x','y','u'}`-vector giving the location of the corner. Thus the row labels of the location matrix are `'x'`, `'y'`, and `'u'`.

The mapping of generalized pixels to colors is given by another matrix, the *color matrix*. Each generalized pixel corresponds to a column of the color matrix, and the label of that column is the pair of pixel coordinates of the top-left corner of that generalized pixel. The column is a `{'r','g','b'}`-vector giving the color of that generalized pixel.

For example, the image  consists of four generalized pixels, comprising a total of nine corners. This image is represented by the location matrix

	(0, 0)	(0, 1)	(0, 2)	(1, 2)	(1, 1)	(1, 0)	(2, 2)	(2, 0)	(2, 1)
x	0	0	0	1	1	1	2	2	2
y	0	1	2	2	1	0	2	0	1
u	1	1	1	1	1	1	1	1	1

and the color matrix

	(0, 0)	(0, 1)	(1, 1)	(1, 0)
b	225	125	75	175
g	225	125	75	175
r	225	125	75	175

By applying a suitable transformation to the location matrix, we can obtain

	(0, 0)	(0, 1)	(0, 2)	(1, 2)	(1, 1)	(1, 0)	(2, 2)	(2, 0)	(2, 1)
x	0	2	4	14	12	10	24	20	22
y	0	10	20	22	12	2	24	4	14

```
u  |      1      1      1      1      1      1      1      1      1
```

which, combined with the unchanged color matrix, looks like this:

4.15.4 *Loading and displaying images*

We provide a module, `image_mat_util`, with some helpful procedures:

- `file2mat`
  - *input:* string giving the pathname of a `.png` image file
  - *output:* a 2-tuple (location matrix, color matrix) representing the image
- `mat2display`
  - *input:* a location matrix and a color matrix (two arguments)
  - *output:* Displays the image in a web browser

Task 4.15.1: Download a `.png` image file, then use `file2mat` to load it and `mat2display` to display it on the screen.

4.15.5 *Linear transformations*

You will be writing a module named `transform` that provides a number of simple linear transformations. Instead of writing procedures that operate on images, your methods will return the transformation matrix and you can apply it to a specific image using matrix multiplication. For each task, we want you to not just write the procedure but also test it on some of the images provided in `matrix_resources/images`.

Task 4.15.2: Write a procedure `identity()` which takes no arguments and returns an identity matrix for location vectors. Verify that this matrix works by applying it first to some points and then to an image, making sure that nothing changes. *(*Hint: *Think about the correct row and column labels.)*

4.15.6 *Translation*

Recall that a translation is a transformation that moves a point (x, y) to $(x + \alpha, y + \beta)$, where α and β are parameters of the transformation. Can you come up with a 2×2 matrix that represents a translation? That is, is there a matrix M such that $x' = Mx$, where x and

x' are the coordinates of a point before and after the translation, respectively? (*Hint:* How does a translation act on the origin, which is the zero vector?)

Now consider a representation of points in two dimensions by 3-dimensional vectors whose third coordinate is fixed to be 1. This is a special case of a representation known as *homogeneous coordinates*. Can you come up with a 3×3 matrix that describes a translation?

Task 4.15.3: Write a procedure `translation(alpha, beta)` that takes two translation parameters and returns the corresponding 3×3 translation matrix. Test it on some images.

4.15.7 *Scaling*

A scaling transformation transforms a point (x, y) into $(\alpha x, \beta y)$, where α and β are the $x-$ and y-scaling parameters, respectively. Can scaling be represented by a 2×2 matrix multiplying a vector $\begin{bmatrix} x \\ y \end{bmatrix}$? Can it be represented by a 3×3 matrix multiplying a vector $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$?

Task 4.15.4: Write a procedure `scale(alpha, beta)` that takes $x-$ and $y-$scaling parameters and returns the corresponding 3×3 scaling matrix that multiplies a vector $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$.

4.15.8 *Rotation*

- What point does the vector $(1, 0, 1)$ represent in homogeneous coordinates? What are the homogeneous coordinates of this point after rotating it about the origin by 30 degrees counterclockwise?
- Answer the same question for the vectors $(0, 1, 1)$ and $(0, 0, 1)$.
- What is the 3×3 matrix M that describes a counterclockwise rotation of 30 degrees about the origin? That is, a matrix M such that $x' = Mx$, where x and x' are the coordinates of a point before and after the rotation, respectively.
- What is the general matrix form for a counterclockwise rotation of θ radians about the origin? Compare this to the 2×2 rotation matrix derived in the book.

Task 4.15.5: Write a procedure `rotation(theta)` that takes an angle in radians and returns the corresponding rotation matrix. *Hint:* Both `sin(·)` and `cos(·)` are available in the `math` module.

4.15.9 *Rotation about a center other than the origin*

Task 4.15.6: Write a procedure `rotation_about(theta, x, y)` that takes three parameters—an angle `theta` in radians, an x coordinate, and a y coordinate—and returns the matrix that rotates counterclockwise about (x, y) by `theta`. *Hint:* Use procedures you've already written.

4.15.10 *Reflection*

A reflection about the y-axis transforms a point (x, y) into a point $(-x, y)$.

Task 4.15.7: Write a procedure `reflect_y()` that takes no parameters and returns the matrix which corresponds to a reflection about the y axis.

Task 4.15.8: Write a procedure `reflect_x()` which that takes no parameters and returns the matrix which corresponds to a reflection about the x axis.

4.15.11 *Color transformations*

Our image representation supports transformations on colors as well as locations. Such a transformation would be applied by multiplying the corresponding matrix times the color matrix.

Task 4.15.9: Write a procedure `scale_color` that takes r, g, and b scaling parameters and returns the corresponding scaling matrix.

Task 4.15.10: Write a procedure `grayscale()` that returns a matrix that converts a color image to a grayscale image. Note that both images are still represented in RGB. If a pixel in the original image had the values r, g, b in each of the color channels, then in the grayscale image it has the value $\frac{77r}{256} + \frac{151g}{256} + \frac{28b}{256}$ in all three color channels.

4.15.12 *Reflection more generally*

Task 4.15.11: Write a procedure `reflect_about(x1,y1,x2,y2)` that takes two points and returns the matrix that reflects about the line defined by the two points. (*Hint:* Use rotations, translations, and a simple reflection).