Exercise

CSC355 Open Source Development

05 October 2020

```
# numpy is a library that gives
# methods for working with arrays,
# vectors, and matrices
import numpy as np

# PIL is the Python Imaging Library
# Pillow is a fork (a copy) of PIL
# Pillow will be our principal example
# of an open source project
from PIL import Image

# TO-DO: Experiment with different values
# for these constants. They determine the
# size of the image that this program draws.
WIDTH = 512
HEIGHT = 512

# Point is a class that models a point in the plane.
class Point:
    # __init__() is the class' constructor.
    def __init__(self, x, y):
        self.x = x
        self.y = y
    # __init__()

    # distance() is a method that computes
    # the Euclidean (as the bird flies) distance
    # between this point and another point
    def distance(self, otherPoint):
        dx = self.x - otherPoint.x
        dy = self.y - otherPoint.y
        return np.sqrt( dx**2 + dy**2 )
    # distance()

    # __str__ is  a method that produces
```

```python
    # string (a printable representation of this point)
    def __str__(self):
        return f'( {self.x:6.2f}, {self.y:6.2f})'
    # __str__()
# Point

# Wave is a class that models a sine wave.
#
# The wave radiates from a point (its center).
#
# The crests of the wave have a height and the
# troughs have a depth. 'Amplitude' is the name
# for the magnitude of the height and depth.
#
# The wavelength is the distance between successive crests.
#
# The phase is a measure of the distance between the center
# and the first crest.
class Wave:
    def __init__(self, center, amplitude, wavelength, phase):
        self.center = center
        self.amplitude = amplitude
        self.wavelength = wavelength
        self.phase = phase
    # __init__()

    # height() is a method that computes the height
    # of the wave at a given point in the plane
    def height(self, point):
        r = point.distance(self.center)
        angle = 2.0 * np.pi * r/self.wavelength + self.phase
        return self.amplitude * np.sin( angle )
    # height()

# Wave

# InterferingWaves is a class that models a collection
# of waves (think of several pebbles tossed into a still
# pond at the same time and how the ripples that spread
# from the points where the stones enter the water will
# collide).
class InterferingWaves:
    # the constructor creates an empty collection
    def __init__(self):
        self.waves = list()
    # __init__()
```

```

    # addWave() is a method for adding a wave to
    # the collection
    def addWave(self, wave):
        self.waves.append( wave )
    # addWave()

    # height() is a method for computing the
    # height of the water at a given point in the
    # plane. This height is the sum of the heights
    # of all of the waves that meet at that point.
    def height(self, point):
        sum = 0.0

        for wave in self.waves:
            sum += wave.height(point)

        return sum
    # height()

# InterferingWaves

class CoordinateSystem:
    # Define a coordinate system by specifying the
    # coordinates of its lower left corner and its
    # upper right corner.
    def __init__(self, xMin, yMin, xMax, yMax):
        self.xMin = xMin
        self.yMin = yMin

        self.xMax = xMax
        self.yMax = yMax
    # __init__()

    # Given a point in this system, produce a new point (x,y)
    # where 0.0 <= x, y <= 1.0.
    # The values of the components of the new point represent
    # fractions of the system's width and height, respectively.
    def normalize(self, point):
        x = (point.x - self.xMin) / (self.xMax - self.xMin)
        y = (point.y - self.yMin) / (self.yMax - self.yMin)

        return Point(x, y)
    # normalize()

    # Given a normalized point (0.0 <= x, y <= 1.0), produce
```

```
    # a new point such that xMin <= x <= xMax and yMin <= y <= yMax.
    def scaleAndTranslate(self, point):
        x = self.xMin + point.x * (self.xMax - self.xMin)
        y = self.yMin + point.y * (self.yMax - self.yMin)

        return Point(x, y)
    # scaleAndTranslate()

# CoordinateSystem

# Transformation models a class that contains
# knowledge of two coordinate systems and the means
# of converting between coordinates given in one
# system and coordinates given in the other system.
class Transform:
    def __init__(self, source, destination):
        self.source = source
        self.destination = destination
    # __init__()

    # map() is a method for making the conversion
    # between coordinates in the source and coordinates
    # in the destination
    def map(self, point):
        n = self.source.normalize( point )

        return self.destination.scaleAndTranslate( n )
    # map()
# Transform

# normalize() is a function for producing a numpy
# array whose elements are all 8 bit unsigned integers
# from a numpy arrays whose elements are all floating
# point values.
def normalize( values ):
    minimum = values.min()
    maximum = values.max()

    fun = lambda x : 256 * (x - minimum) / (maximum - minimum)

    return fun(values)
# normalize()

def main():
    # Print a message just to confirm that the
    # program is working.
```

```
    print( "Guten Tag!" )

    # Create a numpy array of the right size and
    # fill it with zeros.
    amplitudes = np.zeros( (WIDTH, HEIGHT) )

    # Define our world coordinate system and
    # our device coordinate system.
    # The world coordinate system is a system that
    # we choose for our convenience.
    # We will do all of our geometric calculations
    # in the world coordinate system.
    # The device coordiante system corresponds to the
    # window in which the image will appear on the
    # computer's screen.
    world = CoordinateSystem( -1.0, -1.0, +1.0, +1.0 )
    device = CoordinateSystem( 0, 0, WIDTH, HEIGHT )

    device2world = Transform( device, world )


    # Define the waves.

    # TO-DO: Experiment with different values for
    # numberOfWaves, radius, cx, and cy.

    pattern = InterferingWaves()

    numberOfWaves = 4

    radius = 0.4

    cx = 0.0
    cy = 0.0

    for k in range(numberOfWaves):
        angle = 2.0 * np.pi * k / numberOfWaves
        x = cx + radius * np.cos( angle )
        y = cy + radius * np.sin( angle )

        center = Point( x, y )

        # TO-DO: Experiment with different values
        # for amplitude, wavelength, and phase.
        # These are the last 3 arguments of the
        # this constructor.
```

```python
        wave = Wave( center, 1.0, 0.2, 0.0 )

        pattern.addWave( wave )

    # Compute the height of the water
    # at every point in the image.
    for row in range(HEIGHT):
        for column in range(WIDTH):
            u = Point( column, row )
            v = device2world.map( u )

            h = pattern.height( v )
            amplitudes[row, column] = h


    # Normalize heights (that is, express all values on a
    # scale of 0.0 to 1.0), multiply by 256, and convert
    # floating point values to unsigned integers.
    normalizedAmplitudes = normalize(amplitudes).astype(np.uint8)

    print( normalizedAmplitudes.dtype )

    # Create a gray-scale image from the array.
    # TO-DO: Experiment with modes other than "L"
    # and with other algorithms for assigning colors
    # to pixels. You might find this very challenging.
    # I do not expect everyone to complete this task.
    image = Image.fromarray( normalizedAmplitudes, "L" )
    image.show()

# main()

if __name__ == '__main__':
    main()
```