#Run on Jython application

#

# Demonstrates how to create an animation of a 3D sphere using regular points

# on a Display. The sphere is modeled using points on a spherical coordinate

# system (see http://en.wikipedia.org/wiki/Spherical\_coordinate\_system).

# It converts from spherical 3D coordinates to cartesian 2D coordinates (mapping

# the z axis to color). When a point passes the primary meridian (the imaginary

# vertical line closest to the viewer), a note is played based on its latitude

# (low to high pitch). Also the point turns red momentarily.

#

# Based on code by Uri Wilensky (1998), distributed with NetLogo

# under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License.

#

from gui import \*

from music import \*

from random import \*

from math import \*

###### MusicalSphere ######

class MusicalSphere:

"""Creates a revolving sphere that plays music."""

def \_\_init\_\_(self, radius, density, velocity=0.01, frameRate=30, displayTitle="3D Sphere"):

"""

Construct a revolving sphere with given 'radius', 'density'

number of points (all on the surface), moving with 'velocity' angular

(theta / azimuthal) velocity, at 'frameRate' frames (or movements) per

second. Each point plays a note when crossing the zero meridian (the

sphere's meridian (vertical line) closest to the viewer).

"""

# musical parameters

self.instrument = PIANO

self.scale = MAJOR\_SCALE

self.lowPitch = C1

self.highPitch = C6

self.noteDuration = 2000 # milliseconds (2 seconds)

Play.setInstrument(self.instrument, 0) # set the instrument

# visual parameters

self.display = Display(displayTitle, radius\*3, radius\*3) # display to draw sphere

self.display.setColor( Color.BLACK ) # set background color to black

self.radius = radius # how wide circle is

self.numPoints = density # how many points to draw on sphere surface

self.velocity = velocity # how far it rotates per animation frame

self.frameRate = frameRate # how many animation frames to do per second

self.xCenter = self.display.getWidth() / 2 # to place circle at display's center

self.yCenter = self.display.getHeight() / 2

# sphere data structure (parallel lists)

self.points = [] # holds the points

self.thetaValues = [] # holds the points' rotation (azimuthal angle)

self.phiValues = [] # holds the points' latitude (polar angle)

# timer to drive animation

delay = 1000 / frameRate # convert from frame rate to timer delay (in milliseconds)

self.timer = Timer(delay, self.movePoints) # timer to schedule movement

# orange color gradient (used to display depth, the further away, the darker)

black = [0, 0, 0] # RGB values for black

orange = [251, 147, 14] # RGB values for orange

white = [255, 255, 255] # RGB values for white

# create list of gradient colors from black to orange, and from orange to white

# (a total of 25 colors)

self.gradientColors = colorGradient(black, orange, 12) + colorGradient(orange, white, 12) + [white] # remember to include the final color

self.initSphere() # create the circle

def start(self):

"""Starts sphere animation."""

self.timer.start()

def stop(self):

"""Stops sphere animation."""

self.timer.stop()

def initSphere(self):

"""Generate a sphere of 'radius' out of points (placed on the surface of the sphere)."""

for i in range(self.numPoints): # create all the points

# get random spherical coordinates for this point

r = self.radius # all points are placed \*on\* the surface

theta = mapValue( random(), 0.0, 1.0, 0.0, 2\*pi) # random rotation (azimuthal angle)

phi = mapValue( random(), 0.0, 1.0, 0.0, pi) # random latitude (polar angle)

# create a point (with thickness 3) at these x, y coordinates

x, y, color = 0, 0, Color.BLACK

point = Point(x, y, color, 3)

# remember this point and its spherical coordinates (r equals self.radius for all points)

# (append data for this point to the three parallel lists)

self.points.append( point )

self.phiValues.append( phi )

self.thetaValues.append( theta )

# add this point to the display

self.display.add( point )

def sphericalToCartesian(self, r, phi, theta):

"""Convert spherical to cartesian coordinates."""

# adjust rotation so that theta is 0 at max z (i.e., closest to viewer)

x = int( r \* sin(phi) \* cos(theta + pi/2) ) # horizontal axis (pixels are int)

y = int( r \* cos(phi) ) # vertical axis

z = int( r \* sin(phi) \* sin(theta + pi/2) ) # depth axis

# move sphere's center to display's center

x = x + self.xCenter

y = y + self.yCenter

return x, y, z

def depthToColor(self, depth, radius):

"""Map 'depth' to color using the 'gradientColors' RGB colors."""

# create color based on position (points further away have less luminosity)

colorIndex = mapValue(depth, -self.radius, self.radius, 0, len(self.gradientColors)) # map depth to color index

colorRGB = self.gradientColors[colorIndex] # get corresponding RBG value

color = Color(colorRGB[0], colorRGB[1], colorRGB[2]) # and create the color

return color

def movePoints(self):

"""Rotate points on y axis as specified by angular velocity."""

for i in range(self.numPoints):

point = self.points[i] # get this point

theta = (self.thetaValues[i] + self.velocity) % (2\*pi) # increment angle to simulate rotation

phi = self.phiValues[i] # get latitude (altitude)

# convert from spherical to cartesian 3D coordinates

x, y, z = self.sphericalToCartesian(self.radius, phi, theta)

if self.thetaValues[i] > theta: # did we just cross the primary meridian?

color = Color.RED # yes, so sparkle for a split second

pitch = mapScale(phi, 0, pi, self.lowPitch, self.highPitch, self.scale) # phi is latitude

dynamic = randint(0, 127) # random dynamic

Play.note(pitch, 0, self.noteDuration, dynamic) # and play a note (based on latitude)

else: # we are not on the primary meridian, so

# convert depth to color (points further away have less luminosity)

color = self.depthToColor(z, self.radius)

# adjust this point's position and color

self.display.move(point, x, y)

point.setColor(color)

# now, remember this point's new theta coordinate

self.thetaValues[i] = theta

###### MusicalCylinder ######

# This class is inherited from MusicalSphere

class MusicalCylinder(MusicalSphere):

def sphericalToCartesian(self, r, phi, theta):

"""We acctually call cylindicalToCartesian to convert coordinates."""

x, y, z = self.cylindicalToCartesian(self.radius, phi, theta)

return int(x), int(y), int(z)

# You should finish this function

def cylindicalToCartesian(self, r, phi, theta):

"""Convert cylindical to cartesian coordinates.

Convert phi (0 to pi) into y (-r to r).

Calculate x from r and theta.

Calculate z from r and theta.

"""

x, y, z = 0, 0, 0

#phi=randint(-r, r)

phi = mapValue( random(), 0.0, 1.0, -r, r)

# adjust rotation so that theta is 0 at max z (i.e., closest to viewer)

x = int( r \* cos(theta + pi/2) ) # horizontal axis

y = int( phi ) # vertical axis

z = int( r \* sin(theta + pi/2) ) # depth axis

# move sphere's center to display's center

x = x + self.xCenter

y = y + self.yCenter

return x, y, z

#################################################

# create a sphere and start rotating!

#sphere = MusicalSphere(radius=200, density=200, velocity=0.01, frameRate=60)

#sphere.start()

# create a cylinder and start rotating!

cylinder = MusicalCylinder(radius=200, density=200, velocity=0.01, frameRate=60, displayTitle="3D Cylinder")

cylinder.start()