from direct.showbase.ShowBase import ShowBase

from direct.showbase.DirectObject import DirectObject

from direct.gui.DirectGui import \*

from direct.interval.IntervalGlobal import \*

from panda3d.core import GeomVertexFormat, GeomVertexData, Geom, GeomTriangles, GeomVertexWriter,GeomNode, PerspectiveLens, LVector3, LPoint3d, WindowProperties, ClockObject, Thread, LVector3f

from math import cos, pi, sqrt, sin

from array import \*

from panda3d.core import PerspectiveLens, WindowProperties

import sched, time, threading

from direct.task import Task

from datetime import datetime

from panda3d.core import LPlanef, LPoint3f

from panda3d.core import loadPrcFileData

confVars = """

win-size 1280 720

show-frame-rate-meter True

"""

loadPrcFileData("", confVars)

global moveSpeed

global mouseSensitivity

moveSpeed = float(100)

mouseSensitivity = float(50)

global fps

fps = 30

class Sphere:

def \_\_init\_\_(self, x, y, z, r, p, m): #x, y, z are the coordinates of the centre of the sphere, r is the radius, p is the percision (number of points in a semicircle). Minimum value of p is 2. m is the mass

self.velocity = LVector3(0, 0, 0)

self.radius = r

self.mass = m

sphere = list()#this will be a list of lists of points. Each of the lists of points will be one semicircle of the sphere.

for i in range(2 \* p):#each semi circle

semicircle = [LPoint3d(0, 0, 0)] \* (p - 1) #creates a list of points and sets them all to equal zero

for j in range(p - 1): #each point within the semi circle

ang1 = pi \* (i / p)#two component angles of position on sphere

ang2 = (pi / 2) - pi \* ((j + 1) / p)

semicircle[j] = LPoint3d((r \* cos(ang2) \* cos(ang1)), (r \* cos(ang2) \* sin(ang1)), (r \* sin(ang2)))#calculates the x, y, and z values of the point represented by the two angles

sphere.append(semicircle)#adds the semicircle of points to the sphere

snode = GeomNode('sphere')#creates an empty geomnode that will contain the sphere

for i in range(2 \* p):#creates quadrilaterals for every point on the sphere

for j in range(p - 2):

o = i + 1

if (o >= 2 \* p):

o = 0

snode.addGeom(Collision.makeQuadrilateral(self, sphere[i][j], sphere[i][j + 1], sphere[o][j + 1], sphere[o][j]))

top = LPoint3d(0, 0, r)

bottom = LPoint3d(0, 0, -r)

for i in range(2 \* p):

o = i + 1

if (o >= 2 \* p):

o = 0

snode.addGeom(Collision.makeQuadrilateral(self, top, top, sphere[i][0], sphere[o][0]))

snode.addGeom(Collision.makeQuadrilateral(self, bottom, bottom, sphere[i][p - 2], sphere[o][p - 2]))

sphereObject = render.attachNewNode(snode)

sphereObject.setTwoSided(True)

self.sphereModel = sphereObject

self.sphereModel.setPos(x, y, z)

def setVelocity(self, velocityVector):

self.velocity = velocityVector

def updatePosition(self):

global fps

self.sphereModel.setPos(self.sphereModel.getPos() + self.velocity / 100)

def getRadius(self):

return self.radius

def getVelocity(self):

return self.velocity

def getPosition(self):

return self.sphereModel.getPos()

def getMass(self):

return self.mass

class Cubepoopoo:

def \_\_init\_\_(self, x, y, z):

def makeSquare(x1, y1, z1, x2, y2, z2, r, g, b, a):

format = GeomVertexFormat.getV3cp()

vdata = GeomVertexData('square', format, Geom.UHDynamic)

vertex = GeomVertexWriter(vdata, 'vertex')

color = GeomVertexWriter(vdata, 'color')

# make sure we draw the sqaure in the right plane

if x1 != x2:

vertex.addData3(x1, y1, z1)

vertex.addData3(x2, y1, z1)

vertex.addData3(x2, y2, z2)

vertex.addData3(x1, y2, z2)

else:

vertex.addData3(x1, y1, z1)

vertex.addData3(x2, y2, z1)

vertex.addData3(x2, y2, z2)

vertex.addData3(x1, y1, z2)

# adding different colors to the vertex for visibility

color.addData4f(r, g, b, a)

color.addData4f(r, g, b, a)

color.addData4f(r, g, b, a)

color.addData4f(r, g, b, a)

# Quads aren't directly supported by the Geom interface

# you might be interested in the CardMaker class if you are

# interested in rectangle though

tris = GeomTriangles(Geom.UHDynamic)

tris.addVertices(0, 1, 3)

tris.addVertices(1, 2, 3)

square = Geom(vdata)

square.addPrimitive(tris)

return square

# Note: it isn't particularly efficient to make every face as a separate Geom.

# instead, it would be better to create one Geom holding all of the faces.

square0 = makeSquare(0, 0, 0, 0, y, z, 1, 0, 1, 1)

square1 = makeSquare(0, 0, 0, x, 0, z, 1, 1, 0, 1)

square2 = makeSquare(0, 0, 0, x, y, 0, 1, 0, 0, 1)

square3 = makeSquare(x, y, z, x, 0, 0, 1, 0, 1, 1)

square4 = makeSquare(x, y, z, 0, y, 0, 1, 1, 0, 1)

square5 = makeSquare(x, y, z, 0, 0, z, 1, 0, 0, 1)

snode = GeomNode('square')

snode.addGeom(square0)

snode.addGeom(square1)

snode.addGeom(square2)

snode.addGeom(square3)

snode.addGeom(square4)

snode.addGeom(square5)

cube = render.attachNewNode(snode)

cube.setTwoSided(True)

self.cubeModel = cube

self.cubeModel.setPos(0,0,0)

def setPos(self, posX, posY, posZ):

self.cubeModel.setPos(posX,posY,posZ)

def getPosition(self):

return self.cubeModel.getPos()

def setHpr(self, h, p, r):

return self.cubeModel.setHpr(h, p, r)

class Plane:

def \_\_init\_\_(self):

def makeTriangle(x1, y1, z1, x2, y2, z2, x3, y3, z3, r, g, b, a):

format = GeomVertexFormat.getV3cp()

vdata = GeomVertexData('triangle', format, Geom.UHDynamic)

vertex = GeomVertexWriter(vdata, 'vertex')

color = GeomVertexWriter(vdata, 'color')

vertex.addData3(x1, y1, z1)

vertex.addData3(x2, y2, z2)

vertex.addData3(x3, y3, z3)

# adding different colors to the vertex for visibility

color.addData4f(r, g, b, a)

color.addData4f(r, g, b, a)

color.addData4f(r, g, b, a)

color.addData4f(r, g, b, a)

# Quads aren't directly supported by the Geom interface

# you might be interested in the CardMaker class if you are

# interested in rectangle though

tris = GeomTriangles(Geom.UHDynamic)

tris.addVertices(0, 1, 3)

tris.addVertices(1, 2, 3)

triangle = Geom(vdata)

triangle.addPrimitive(tris)

return triangle

# Note: it isn't particularly efficient to make every face as a separate Geom.

# instead, it would be better to create one Geom holding all of the faces.

x0 = 0

y0 = 0

z0 = 0

x1 = 1000

y1 = 1000

z1 = 0

x2 = 0

y2 = 1000

z2 = 1000

# 3 points of plane

point0 = LPoint3f(x0, y0, z0)

point1 = LPoint3f(x1, y1, z1)

point2 = LPoint3f(x2, y2, z2)

#displaying the verticies with spheres

sPoint0 = Sphere(x0, y0, z0, 3, 5, 5)

sPoint1 = Sphere(x1, y1, z1, 3, 5, 5)

sPoint2 = Sphere(x2, y2, z2, 3, 5, 5)

self.plane0 = LPlanef(point0, point1, point2)

normal = self.plane0.getNormal()

print("normal = ",normal)

planeTriangle = makeTriangle(point0.getX(), point0.getY(), point0.getZ(), point1.getX(), point1.getY(), point1.getZ(), point2.getX(), point2.getY(), point2.getZ(), 1, 1, 1, 1)

snode = GeomNode('square')

snode.addGeom(planeTriangle)

plane = render.attachNewNode(snode)

plane.setTwoSided(True)

self.planeModel = plane

def disToPlane(self, point):

return self.plane0.distToPlane(point)

def getClosePoint(self, point):

return self.plane0.project(point)

class Collision(ShowBase):

def \_\_init\_\_(self):

ShowBase.\_\_init\_\_(self)

lens = PerspectiveLens()

base.setFrameRateMeter(True)

Collision.movement(self)

self.scene = self.loader.loadModel("models/environment")

self.scene.reparentTo(self.render)

self.scene.setScale(0.25, 0.25, 0.25)

self.scene.setPos(-8, 42, 0)

global t, curTime, sphere1, sphere0, sphere2, sphere3, cube0, plane0

sphere0 = Sphere(20, 2, 20, 6, 8, 1)

sphere1 = Sphere(-20, 0, 22, 2, 8, 1)

sphere2 = Sphere(10, 2, 20, 2, 8, 1)

#plane collision sphere

sphere3 = Sphere(100, 2, 20, 4, 8, 1)

plane0 = Plane()

cube0 = Cubepoopoo(10, 1000, 1000)

cube0.setHpr(0, 0, -20)

cube0.setPos(0, -500, -500)

sphere0.setVelocity(LVector3(-10, 0, 0))

sphere1.setVelocity(LVector3(-1000, 0, 0))

sphere2.setVelocity(LVector3(0, 3, 5))

curTime = datetime.now()

global fps

self.taskMgr.doMethodLater(1/fps, self.physicsUpdate, 'physics')

print(plane0.disToPlane(sphere3.getPosition()))

def physicsUpdate(self, task):

#sphere collisions

distance = self.getDistance(sphere0, sphere1)

if(distance < sphere0.getRadius() + sphere1.getRadius()):

vecx = sphere0.getPosition() - sphere1.getPosition()

vecx.normalize()

vecv0 = sphere0.getVelocity()

x0 = vecv0.getX()

vecv0x = vecx \* x0

vecv0y = vecv0 - vecv0x

m0 = sphere0.getMass()

vecx \*= -1

vecv1 = sphere1.getVelocity()

x1 = vecv1.getX()

vecv1x = vecx \* x1

vecv1y = vecv1 - vecv1x

m1 = sphere1.getMass()

sphere0.setVelocity(vecv0x \* (m0-m1) / (m0+m1) + vecv1x \* (2 \* m1) / (m0 + m1) + vecv0y)

sphere1.setVelocity(vecv0x \* (2 \* m0) / (m0+m1) + vecv1x \* (m1-m0) / (m0+m1) + vecv1y)

sphere0.updatePosition()

sphere1.updatePosition()

#wall collision

wallDistance = cube0.getPosition().getX() - sphere2.getPosition().getX()

#plane collsion

if (plane0.disToPlane(sphere0.getPosition()) <= sphere0.getRadius() and plane0.disToPlane(sphere0.getPosition()) >= -sphere0.getRadius()):

vel = (sphere0.getVelocity().project(plane0.getClosePoint(sphere0.getPosition()) - sphere0.getPosition()))

sphere0.setVelocity(sphere0.getVelocity() - vel - vel)

print(plane0.disToPlane(sphere0.getPosition()))

#print(wallDistance)

if(wallDistance < sphere2.getRadius()):

xSphere = sphere2.getVelocity().get\_x()

ySphere = sphere2.getVelocity().get\_y()

zSphere = sphere2.getVelocity().get\_z()

sphere2.setVelocity(LVector3(-xSphere, ySphere, zSphere))

sphere2.updatePosition()

return task.again

def getDistance(self, S0, S1):

between = S0.getPosition() - S1.getPosition()

distance = sqrt(between.getX() \*\* 2 + between.getY() \*\* 2 + between.getZ() \*\* 2)

return distance

def movement(self):

self.xray\_mode = False

self.show\_model\_bounds = False

base.disableMouse() #disables default mouse control

props = WindowProperties()

props.setCursorHidden(True) #hides the cursor

base.win.requestProperties(props)

# Setup controls

self.keys = {}

for key in ['a', 'd', 'w', 's', 'c', 'space']:

self.keys[key] = 0 #array that stores the state of the above keys (1 is pressed down, 0 is not)

self.accept(key, self.push\_key, [key, 1])

self.accept('shift-%s' % key, self.push\_key, [key, 1]) #if the key is pressed or the key is pressed with shift, it will be registered

self.accept('%s-up' % key, self.push\_key, [key, 0])

self.accept('escape', \_\_import\_\_('sys').exit, [0]) #closes program if escape is pressed

# Setup camera

self.lens = PerspectiveLens()

self.lens.setFov(60)

self.lens.setNear(0.01)

self.lens.setFar(1000.0)

#self.cam.node().setLens(self.lens)

self.heading = 0.0

self.pitch = 0.0

self.taskMgr.add(self.update, 'main loop')

def push\_key(self, key, value):

self.keys[key] = value

def update(self, task):

mw = base.mouseWatcherNode

x = 0

y = 0

if mw.hasMouse():

# get the position relative to centre

x, y = mw.getMouseX(), mw.getMouseY()

# move mouse back to center

props = base.win.getProperties()

base.win.movePointer(0, props.getXSize() // 2, props.getYSize() // 2)

delta = globalClock.getDt()

move\_x = delta \* moveSpeed \* self.keys['d'] - delta \* moveSpeed \* self.keys['a']

move\_z = delta \* moveSpeed \* self.keys['w'] - delta \* moveSpeed \* self.keys['s']

move\_y = delta \* moveSpeed \* self.keys['space'] - delta \* moveSpeed \* self.keys['c']

self.camera.setPos(self.camera, move\_x, move\_z, move\_y)

self.heading += (-x \* mouseSensitivity)

if (self.pitch + y \* mouseSensitivity > 90):

self.pitch = 90

elif (self.pitch + y \* mouseSensitivity < -90):

self.pitch = -90

else:

self.pitch += (y \* mouseSensitivity)

self.camera.setHpr(self.heading, self.pitch, 0)

return task.cont

def makeQuadrilateral(self, point1, point2, point3, point4): #input the four points (LPoint3d) that a quadrilateral will be drawn between. Ensure that the four points make a U shape if you were to draw a line between them (Not an N or X shape)

#separate the coordinates from all 4 points

x1 = point1.getX()

x2 = point2.getX()

x3 = point3.getX()

x4 = point4.getX()

y1 = point1.getY()

y2 = point2.getY()

y3 = point3.getY()

y4 = point4.getY()

z1 = point1.getZ()

z2 = point2.getZ()

z3 = point3.getZ()

z4 = point4.getZ()

format = GeomVertexFormat.getV3cp() #format contains vertex location and colour of the vertex

vdata = GeomVertexData('square', format, Geom.UHDynamic)

vertex = GeomVertexWriter(vdata, 'vertex')#writers for the vertex and the colour

color = GeomVertexWriter(vdata, 'color')

vertex.addData3(x1, y1, z1) #adds the position of the four vertexes

vertex.addData3(x2, y2, z2)

vertex.addData3(x3, y3, z3)

vertex.addData3(x4, y4, z4)

# adding different colors to the vertex for visibility. These colours are expressed in RGBA.

color.addData4f(0, 0, 1, 1)

color.addData4f(0, 0, 1, 1)

color.addData4f(0, 0.5, 1, 1)

color.addData4f(0.5, 0, 1, 1)

tris = GeomTriangles(Geom.UHDynamic) #creates two triangles to represent the quadrilateral

tris.addVertices(0, 1, 3)

tris.addVertices(1, 2, 3)

square = Geom(vdata)

square.addPrimitive(tris)#combines the triangles into one quadrilateral

return square

t = Collision()

t.run()