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, math

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(1000)

mouseSensitivity = float(50)

global fps

fps = 120

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)

self.points = []

self.points.append(LPoint3f(0,0,0))

self.points.append(LPoint3f(0, y, z))

self.points.append(LPoint3f(x, 0, z))

self.points.append(LPoint3f(x, y, 0))

self.points.append(LPoint3f(x, y, z))

self.points.append(LPoint3f(x, 0, 0))

self.points.append(LPoint3f(0, y, 0))

self.points.append(LPoint3f(0, 0, z))

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)

self.mass = 10

self.rotationalVelocity = (0,0,0)

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

self.gravity = LVector3(0, 0, -100)

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

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

def setPos(self, Pos):

self.cubeModel.setPos(Pos)

def updatePosition(self):

global fps

self.cubeModel.setPos(self.cubeModel.getPos() + self.velocity / fps)

for i in range(8):

self.points[i] += self.velocity / fps

x = (self.points[i] - self.cubeModel.getPos())[0]

y = (self.points[i] - self.cubeModel.getPos())[1]

z = (self.points[i] - self.cubeModel.getPos())[2]

h = self.rotationalVelocity[0] / fps / 180 \* pi

p = self.rotationalVelocity[1] / fps / 180 \* pi

r = self.rotationalVelocity[2] / fps / 180 \* pi

x1 = x \* cos(h) - y \* sin(h)

y1 = x \* sin(h) + y \* cos(h)

z1 = z

x2 = x1

y2 = y1 \* cos(p) - z1 \* sin(p)

z2 = y1 \* sin(p) + z1 \* cos(p)

x3 = z2 \* sin(r) + x2 \* cos(r)

y3 = y2

z3 = z2 \* cos(r) - x2 \* sin(r)

self.points[i] = LPoint3f(x3, y3, z3) + self.cubeModel.getPos()

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 = 0

z1 = 0

x2 = 0

y2 = -1000

z2 = 1000

self.cof = 1.5

self.angle = pi/4

# 3 points of plane

point0 = LPoint3f(x0, y0, z0)

point1 = LPoint3f(x1, y1, z1)

point2 = LPoint3f(x2, y2, z2)

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

self.normal = self.plane0.getNormal()

print("normal = ", self.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, cube0, plane0, fps, timeSinCol, checkBounce

#plane collision sphere

plane0 = Plane()

cube0 = Cubepoopoo(30, 30, 30)

cube0.setHpr(0, 45, 0)

startPos = (100, -600, 700)

cube0.setPos(startPos)

for x in cube0.points:

x += (startPos)

cube0.velocity = LVector3(100, 0, 0)

timeSinCol = 10000

checkBounce = True

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

def physicsUpdate(self, task):

global timeSinCol, projNor, checkBounce

projNor = (0,0,0)

cube0.acceration = LVector3(0,0,0)

fn = 0

ff = 0

colPoints = []

timeSinCol += 1

for x in cube0.points:

disToPlane = plane0.disToPlane(x)

if (disToPlane > -0.25 and disToPlane < 0.25):

colPoints.append(x)

if (cube0.velocity.project(plane0.normal).length() < 1):

cube0.velocity -= cube0.velocity.project(plane0.normal)

checkBounce = False

else:

checkBounce = True

if (checkBounce == True and timeSinCol > 1):

timeSinCol = 0

print("collided")

projNormal = -cube0.velocity.project(plane0.normal)

#print(projNormal)

planeVec = plane0.getClosePoint(x + cube0.velocity) - x

projPlane = cube0.velocity.project(planeVec)

cube0.velocity = projNormal/3 + projPlane

# Fn = -cos(45)(Fg)

# Fn = -cos(45)(m)(g)

fn = -plane0.normal.normalized() \* cube0.mass \* cube0.gravity.length() \* math.cos(plane0.angle)

# #ff = cof(fn)

ff = -cube0.velocity.normalized() \* fn.length() \* plane0.cof

# projNor = cube0.velocity.project(plane0.normal)

if (checkBounce == False):

# Fn = -cos(45)(Fg)

# Fn = -cos(45)(m)(g)

fn = -plane0.normal.normalized() \* cube0.mass \* cube0.gravity.length() \* math.cos(plane0.angle)

# #ff = cof(fn)

ff = -cube0.velocity.normalized() \* fn.length() \* plane0.cof

#a = g + fn/m + ff/m

cube0.acceration = cube0.acceration + cube0.gravity + fn/cube0.mass + ff/cube0.mass

#v = v + at

cube0.velocity = cube0.velocity + (cube0.acceration \* 1/fps)

print(cube0.velocity)

cube0.updatePosition()

return task.again

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

t = Collision()

t.run()