ФИНКИ

Проект по предметот Основи на компјутерска графика

Тема:

Соларен систем со две Сонца

Изработено од студентите:

Елена Папазова 176011

Софија Симјановска 175003

Искористен програм:

Pybullet Physics Engine

Гравитација

Гравитацијата претставува универзална сила на привлечност која делува помеѓу целата материја. Таа е најслабата сила во природата и затоа не игра улога во одредувањето на внатрешните својства на секојдневната материја. Од друга страна, преку нејзиното далечно достигнување и универзално дејствување, ги контролира траекториите (патеките) на телата во соларните системи и насекаде во Универзумот, и ги контролира структурите и еволуцијата на ѕвездите, галаксиите и целиот Космос. На Земјата сите тела имаат тежина, односно надолна сила на гравитација пропорционална на нивната маса. Гравитацијата на Земјата (и општо на сите небесни тела) се мери со забрзувањето што го постигнуваат телата (објектите) што слободно паѓаат.

Работата и делата на Исак Њутн (Класичната теорија на гравитационата сила) и Алберт Ајнштајн (Теорија на релативноста) доминираат во развојот на делот од физиката за гравитација и гравитационата теорија. Њутн ја открил врската помеѓу дижењето на Месечината и движењето на тело што слободно паѓа на Земјата. Со своите теории за динамика и гравитација, тој ги објаснил Кеплеровите закони и ја поставил основал модерната квантитативна наука за гравитација. Њутн го претпоставил постоењето на привлечна сила помеѓу сите масивни тела, таква што не бара телесен контакт и делува на на далечина. Осврнувајќи се на неговиот закон за инерција (тела на кои не дејствува сила се движат со константна брзина по права линија), Њутн заклучил дека потребна е сила од страна на Земјата да дејствува врз Месечината за да таа кружи околу неа. Тој сфатил дека оваа сила, на голема далечина, би била истата сила со која Земјата ги привлекува објектите да паднат на нејзината површина.

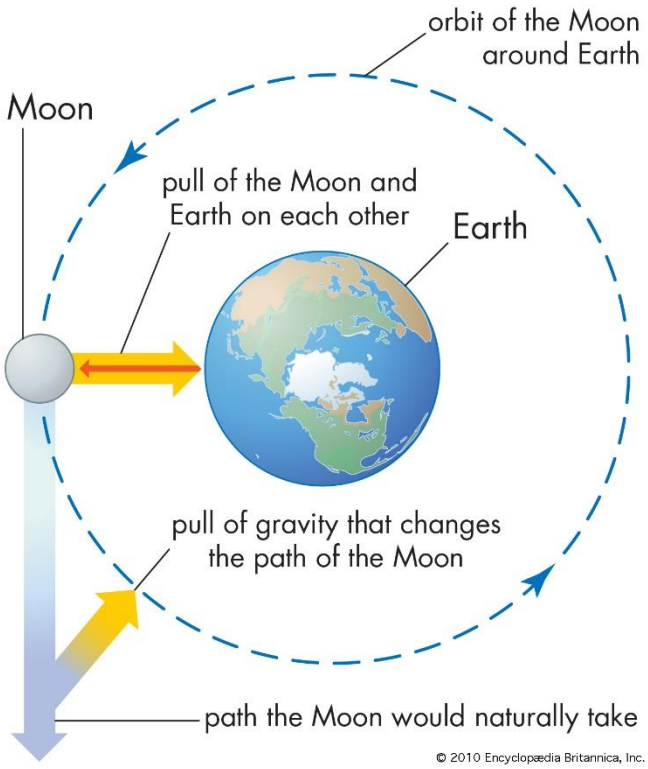
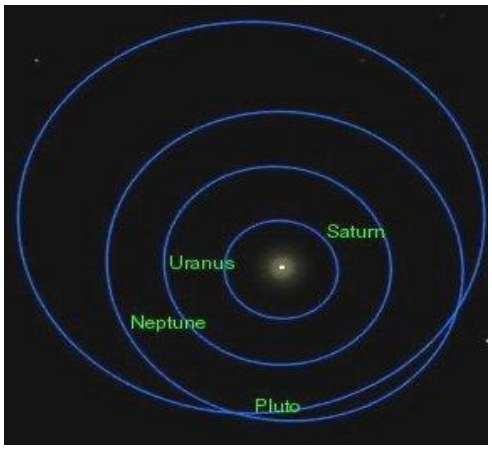


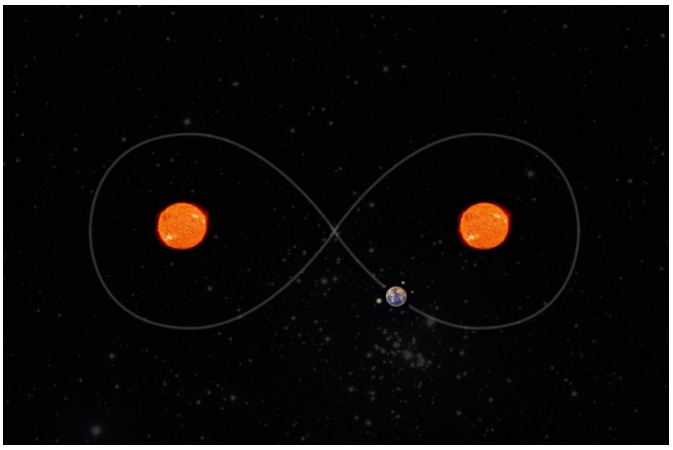
Figure-8 Орбиталната Патека и што таа открива

Во Космосот знаеме дека постојат многу различни форми на орбити на телата.

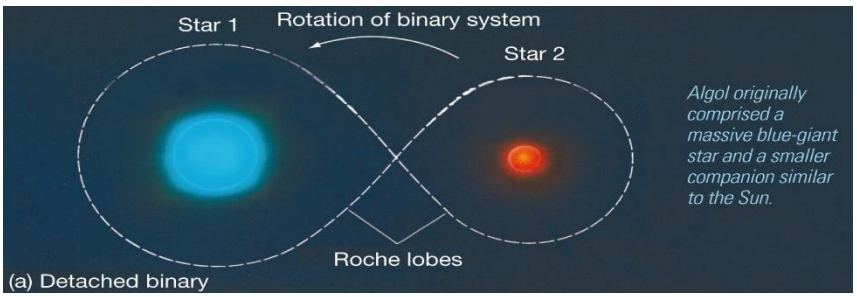
Познати орбитални облици:

- Кружна орбита – Орбитата може да биде во форма на круг, каде телото кое се движи по таа патека се наоѓа на константна оддалеченост од телото околу кое кружи.
- Елиптична орбита – Кај овој тип на орбита, дистанцата на телото што орбитира варира преку целата патека долж орбитата околу другото небесно тело.
- Издолжена елиптична орбита – Како и кај елиптичната орбита, и тука дистанцата на орбитирачкото тело спрема телото околу кое се движи варира преку целата патека, со тоа што тука е позначително нагласена таа разлика.





Посебен и редок случај е орбитата во форма на бројот 8. За да се постигне таквата патека на движење на телото, мора да има две други масивно големи небесни тела (ѕвезди), со приближно еднаква гравитациона сила, на еднаква оддалеченост од тоа тело. Исто така, потребно е и самите тие ѕвезди да се на доволно големо меѓусебно растојание. Ако двете ѕвезди во тој *Бинарен ѕвезден систем* се на поблиско растојание, тогаш гасот и друга материја од едната ѕвезда ќе почне да се пренесува на другата (поголема) ѕвезда. Како што самите ѕвезди еволуираат со тек на времето, така и овој Бинарен свезден систем еволуира и може да настане тоа пренесување на материја од една ѕвезда на друга. Ваквиот систем е познат под името *Алгол систем*.



Документација за кодот

Во оваа проектна, преку користење на Pybullet Physics Engine кој овозможува реална симулација на физика и движење на роботи, ќе го претставиме движењето на една планета околу две ѕвезди.

```
import pybullet
```

Најпрво се поврзуваме со серверот кој ги овозможува симулациите:

```
physicsClient = pybullet.connect(pybullet.GUI)
pybullet.setAdditionalSearchPath(pybullet_data.getDataPath())
```

Потоа се мести камерата за симулацијата:

```
pybullet.resetDebugVisualizerCamera(cameraDistance = 2,
                                    cameraYaw = 10,
                                    cameraPitch = 5,
                                    cameraTargetPosition = [0, -4, 1])
```

Креираме колизиски објекти – сфери со соодветни радиуси:

```
sphere = pybullet.createCollisionShape(pybullet.GEOM_SPHERE, radius = 0.6)
sphere2 = pybullet.createCollisionShape(pybullet.GEOM_SPHERE, radius = 1)
sphere3 = pybullet.createCollisionShape(pybullet.GEOM_SPHERE, radius = 0.2)
```

- 0.6 за Sun, 1 за Neutron и 0.2 за планетата соодветно со нивните позиции на координатната оска.

```
- spherePos1 = [2,-2,0.2]
  spherePos2 = [-2.5,-1,2.5]
  spherePos3 = [0,0,0]
```

Колизиските објекти ги креираме сега со VisualShape:

```
visualShapeId = pybullet.createVisualShape(shapeType = pybullet.GEOM_SPHERE,
                                rgbaColor = [1,0.47,0.97,1],
                                radius = 0.2
                                  )
visualShapeId2 = pybullet.createVisualShape(shapeType = pybullet.GEOM_SPHERE,
                                rgbaColor = [0.26,0.8,0.9,1],
                                radius = 1
                                  )
visualShapeId3 = pybullet.createVisualShape(shapeType = pybullet.GEOM_SPHERE,
                                rgbaColor = [1,0.7,0.1,1],
```

```
                                radius = 0.5
                                  )
```

Во кои ID-ата понатаму ке ги препратиме за создавање на Multibody.

Ги креираме со класата GravityMovement за полесна манипулација со нив:

```
planet = GravityMovement(mass, sphere3, visualShapeId, spherePos3, planetGravity)
Neutron = GravityMovement(mass, sphere2, visualShapeId2, spherePos2, neutronGravity)
Sun = GravityMovement(mass, sphere, visualShapeId3, spherePos1, sunGravity)
```

Во GravityMovement при иницијализација ги земаме атрибутите и дополнително пресметуваме и чуваме радиус на гравитација, позиција, тежина и ја креираме сферата со овие атрибути.

```
def __init__(self, mass, sphere, id, position, gravity):
    self.position = position
    self.gravity = gravity
    self.gravityPosition = [x + self.gravity for x in position]
    self.mass = mass
    self.sphere = sphere
    self.id = id
    self.r_hor = self.gravity
    self.sphereUid = pybullet.createMultiBody(self.mass,
                                self.sphere,
                                self.id,
                                self.position,
                                )
```

Во класава ги имаме функциите за движење околу двата објекти испишани со координати:

```
def moveUp(self):
    self.position = [self.position[0] - 0.005, self.position[1] + 0.002, self.position[2] + 0.005]
    self.setGrav(self.position)
    pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())
def moveCurvyUp(self):
    self.position = [self.position[0] + 0.005, self.position[1] - 0.005, self.position[2] + 0.004]
    self.setGrav(self.position)
    pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())
def moveDown(self):
    self.position = [self.position[0] + 0.006, self.position[1] + 0.00005, self.position[2] - 0.005]
    self.setGrav(self.position)
    pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())
def moveCurvyBack(self):
    self.position = [self.position[0] - 0.005, self.position[1] - 0.001, self.position[2] + 0.004]
    self.setGrav(self.position)
    pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())
def moveCurvyDown(self):
```

```
    self.position = [self.position[0] - 0.005, self.position[1] - 0.005, self.position[2] - 0.007]
    self.setGrav(self.position)
    pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())

def moveCurvyForward(self):
    self.position = [self.position[0] + 0.006, self.position[1] + 0.005, self.position[2] - 0.001]
    self.setGrav(self.position)
    pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())
def moveCurvyUpward(self):
    self.position = [self.position[0] - 0.002, self.position[1] - 0.0085, self.position[2] - 0.001]
    self.setGrav(self.position)
    pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())
def moveCurvyBackwards(self):
    self.position = [self.position[0] - 0.003, self.position[1] + 0.002, self.position[2] + 0.002]
    self.setGrav(self.position)
    pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())
def moveToDefault(self):
    if not math.fabs(self.position[0]) < 0.004:
        if self.position[0] < 0:
            self.position[0] = self.position[0] + 0.004
        elif self.position[0] > 0:
            self.position[0] = self.position[0] - 0.004
    else:
        self.position[0] = 0
    if not math.fabs(self.position[1]) < 0.008:
        if self.position[1] < 0:
            self.position[1] = self.position[1] + 0.008
        elif self.position[1] > 0:
            self.position[1] = self.position[1] - 0.008
    else:
        self.position[1] = 0
    if not math.fabs(self.position[2]) < 0.0012:
        if self.position[2] < 0:
            self.position[2] = self.position[2] + 0.0012
        elif self.position[2] > 0:
            self.position[2] = self.position[2] - 0.0012
    else:
        self.position[2] = 0
    self.setGrav(self.position)
    pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())

def moveX(self):
    self.position = [self.position[0] + 0.007, self.position[1] + 0.002, self.position[2] - 0.0001]
    self.setGrav(self.position)
    pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())

def movecurvyAround(self):
    self.position = [self.position[0] + 0.0031, self.position[1] - 0.006, self.position[2] - 0.002]
    self.setGrav(self.position)
    pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())
```

```
def moveY(self):
    self.position = [self.position[0] - 0.004, self.position[1], self.position[2] - 0.002]
    self.setGrav(self.position)
    pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())
```

Дополнителна надворешна фукнција за пресметување на растојание:

```
def distance(centerPos1, centerPos2, r1, r2):
    p1 = numpy.array(centerPos1); p2 = numpy.array(centerPos2)
    sq_dist = numpy.sum((p1 - p2)**2, axis =0)
    dist = numpy.sqrt(sq_dist)
    return dist - r1 - r2
```

Во главниот циклус – step симулацијата – имаме:

```
pybullet.stepSimulation()
time.sleep(1./240.)
```

за да ја започне симулацијата.

Во главниот циклус се пресметува со flags каде се наоѓа во чиво гравитациско поле е моментално и Distance растојанието меѓу нив:

```
if neutronFlag == 1 and sunFlag == 0:
    Distance = distance(planet.getCenter(), Neutron.getCenter(), planet.getGravityField(),
Neutron.getGravityField())
    if Distance > 0:
        if planet.position[0] < 0 and planet.position[1] < 0 and planet.position[2] < 0:
            planet.moveToDefault()
        else:
            planet.moveUp()
        print("MOVING UP NOT YET IN GRAVITY")
```

Понатаму има пресметки за кружење околу Neutron се додека не излезе од полето:

```
else:
    print("CAUGHT GRAVITY")
    if planet.position[2] <= Neutron.position[2] and planet.position[0] >= Neutron.position[0]:
        print("MOVING CURVY UP")
        planet.moveCurvyUp()
    elif planet.position[0] >= Neutron.position[0]and planet.position[2] >=
Neutron.position[2]:
        print("MOVING CURVY BACK")
        planet.moveCurvyBack()
    elif planet.position[0] <= Neutron.position[0] and planet.position[2] >=
Neutron.position[2]:
        print("MOVING CURVY DOWN")
```

```
        planet.moveCurvyDown()
    elif planet.position[0] <= Neutron.position[0] and planet.position[2] <=
Neutron.position[2]:
        print("FINAL MOVEMENT BEFORE EXITING DOWN")
        planet.moveDown()
        neutronFlag = 0
```

Надвор од полето на Neutron:

```
if neutronFlag == 0 and sunFlag == 0:
    print("EXITING GRAVITY")
    if planet.position[0] <= Neutron.position[0] and planet.position[2] <= Neutron.position[2]:
        print("STILL BEHIND THE NEUTRON ON THE WAY OUT")
        planet.moveDown()
    else:
        print("ON ITS WAY OUT TO SUN GRAVITY")
        planet.moveX()
        sunFlag = 1
```

Кога ке влезе во Sun’s gravity се прават истите пресметки:

```
if sunFlag == 1 and neutronFlag == 0:
    print("ON THE SUN'S GRAVITY")
    Distance = distance(planet.getCenter(), Sun.getCenter(), planet.getGravityField(),
Sun.getGravityField())
    if Distance > 0:
        print("NOT YET CAUGHT BY SUN")
        planet.moveX()
```

Каде понатаму го има стандардното движење околу:

```
else:
    if  planet.position[2] >= Sun.position[2] and planet.position[0] <= Sun.position[0]:
        print("MOVING CURVY FORWARD AROUND THE SUN")
        planet.moveCurvyForward() #*****
    elif planet.position[0] >= Sun.position[0] and planet.position[2] >= Sun.position[2]:
        print("THE SWOOP")
        planet.movecurvyAround() #**************
    elif planet.position[0] >= Sun.position[0] and planet.position[2] <= Sun.position[2]:
        print("MOVING AROUND BACK")
        planet.moveCurvyUpward()  # ****
        if planet.position[1] < Sun.position[1]:
            planet.moveY()
    elif planet.position[2] <= Sun.position[2] and planet.position[0] <= Sun.position[0]:
        print("MOVING UP BACKWARDS")
        planet.moveCurvyBackwards()
       # sunFlag = 0
        neutronFlag = 1
```

На крај при излегување од полето на Sun се придвижува кон координатниот почеток назад:

```
if sunFlag == 1 and neutronFlag == 1:
    if planet.position[2] < 0 or planet.position[0] > 0 or planet.position[1] < 0:
        planet.moveToDefault()
    else:
        sunFlag = 0
```

За крај секогаш треба да се одјавиме од серверот:

```
pybullet.disconnect()
```