Lab06. Animation

2 Moving Object

- Lab06-1 : Moving with time

```
from OpenGL.GLUT import *
from OpenGL.GL import *
from OpenGL.GLU import *
import time
import numpy as np


dt = -1
currentTime = 0
lastTime=0

def TimerOn() :
    if dt>0 :
        return True
    else :
        return False

def TimerStart():
    global currentTime, lastTime, dt
    if dt<0 :
        currentTime = time.perf_counter()
        lastTime = currentTime

def TimerGetDt():
    global currentTime, lastTime, dt
    currentTime = time.perf_counter()
    dt = currentTime - lastTime
    lastTime = currentTime
    return dt

ball1 = np.array([0,0,0])
ball2 = np.array([0,0,5])

ball1V = np.array([1,0,0])
ball2V = np.array([0.5, 0,0])

simulationStart = False

def GLinit() :
    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
    glutInitWindowSize(500, 500)
    glutInitWindowPosition(100, 100)
    glutCreateWindow(b"Lab06-1:Moving-ball")

def RegisterCallbacks() :
    glutDisplayFunc(draw)
    glutIdleFunc(draw)
    glutKeyboardFunc(key)
```

```
def key(k, x,y) : #To move ball, hit any key.
    global simulationStart
    simulationStart = True

def drawLine(x,y,z, xx,yy,zz) :
    glBegin(GL_LINES)
    glVertex3f(x,y,z)
    glVertex3f(xx,yy,zz)
    glEnd()

def drawBall(pos) :
    glPushMatrix()
    glTranslatef(pos[0], pos[1], pos[2])
    glutWireSphere(1.0, 10,10)
    glPopMatrix()

def draw():
    global ball1, ball2

    if TimerOn() != True :
        TimerStart()

    dt = TimerGetDt()

    glClear(GL_COLOR_BUFFER_BIT)

    # Lens
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(60, 1.0, 0.1, 1000.0)

    # World
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    gluLookAt(20,20,20, 0,0,0, 0,1,0)

    glColor3f(1,0,0)
    drawLine(0, 0, 0, 10, 0, 0)
    glColor3f(0, 1, 0)
    drawLine(0, 0, 0,  0,10, 0)
    glColor3f(0, 0, 1)
    drawLine(0, 0, 0,  0, 0,10)

    glColor3f(1,1,1)

    if simulationStart :
        ball1 = ball1 + ball1V * dt
        ball2 = ball2 + ball2V * dt

    drawBall(ball1)
    drawBall(ball2)
    glFlush()

GLinit()
RegisterCallbacks()
glutMainLoop()

# End of program
```

Moving object with Kinematics

- 움직이는 방식의 변화 : 운동학 기반의 함수에 의해 위치 계산
  - P(t) : t 시간의 위치
  - P(t) = P(0) + V(0)*t + 1/2 * g * t^2
  - Ball = Ball_최초 + 속도_최초 * t + 0.5 * 중력가속도 * t*t

```
ball1 = np.array([0,0,0])
ball2 = np.array([0,0,5])

ball1V = np.array([1,0,0])
ball2V = np.array([0.5, 0,0])

g = np.array([0, -10, 0])
et = 0.0 # elapsed time = dt의 누적

simulationStart = False

def GLinit() :
```

4 Moving object with Kinematics(2)

- In draw() :

```
def draw():
    global ball1, ball2, ball1V, ball2V, g, dt, et

    if TimerOn() != True :
        TimerStart()
        et = 0.0
        dt = 0.0

    #dt = TimerGetDt()
    glClear(GL_COLOR_BUFFER_BIT)

    # Lens
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(60, 1.0, 0.1, 1000.0)

    # World
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    gluLookAt(20,20,20, 0,0,0, 0,1,0)

    glColor3f(1,0,0)
    drawLine(0, 0, 0, 10, 0, 0)
    glColor3f(0, 1, 0)
    drawLine(0, 0, 0,  0,10, 0)
    glColor3f(0, 0, 1)
    drawLine(0, 0, 0,  0, 0,10)

    glColor3f(1,1,1)

    if simulationStart :
        et = et + dt
        P1 = ball1 + ball1V * et + 0.5 * g * et**2
        P2 = ball2 + ball2V * et + 0.5 * g * et**2
        drawBall(P1)
        drawBall(P2)
    else:
        drawBall(ball1)
        drawBall(ball2)
    glFlush()

    dt = TimerGetDt()
```

운동학 함수에 의해 위치 계산

Class for Simulation

- Package : CGGame (CGGame.py)
- Classes
  - Graphics : 그리기 위한 함수
    - drawGrid(), drawBall()
  - Timer : 시간 제어를 위한 함수
    - 초기화__init__(), isTimerOn(), start(), stop(), getDt(), getEt()
  - Camera : 카메라 설정에 관한 함수
    - 초기화(__init__()), applyCamera(), setLens(), setPos(), setAsp()
  - Game : Animation에 관한 함수
    - 초기화(__init__(), reshape(), grid(), timeStart(), timeStop(), getDt(), getEt(), frame(), afterFrame(), start(), drawBall()

Moving object with CGGame

```
from OpenGL.GLUT import *
from OpenGL.GL import *
from OpenGL.GLU import *
import time
import numpy as np

import CGGame

from math import *


class myGame(CGGame.Game) :

    def frame(self):
        dt = self.getDt()
        et = self.getEt()

        super().frame()
        # your code here
        self.drawBall([5.0*cos(et),0, 5.*sin(et)])
        super().afterFrame()

game = myGame(1500,1000, b"Lab06-3:moving ball with CGGame")
game.grid(True)

def key(k, x,y) :
    game.timerStart()
def draw() :
    game.frame()

game.start(draw, key)
```

Moving object with kinematics using CGGame

- Package를 이용하여 동력학적인 물체의 움직임 시뮬레이션

```
import time
import numpy as np
import CGGame
from math import *

class myGame(CGGame.Game) :

    def __init__(self, w, h, title):
        super().__init__(w,h,title)
        self.loc = np.array([0.,  0. , 0.])
        self.vel = np.array([0., 10. , 0.])
        self.acc = np.array([0., -9.8, 0.])

    def frame(self):
        dt = self.getDt()
        et = self.getEt()
        super().frame()
        # your code here
        # 가속을 수치적분한다!!!
        self.vel += self.acc * dt
        # 속도를 수치적분한다!!!
        self.loc += self.vel * dt
        self.drawBall(self.loc)
        super().afterFrame()
```

```
game = myGame(1500,1000, b"Lab06-4:Moving Ball
with Dynamics")
game.grid(True)

def key(k, x,y) :
    game.timerStart()
def draw() :
    game.frame()

game.start(draw, key)
```

8 실습문제

1. Lab06-1에서 공의 색깔을 다른 구로 표현해보자
2. Lab06-2을 이용하여 하나의 공은 x축을 중심으로 원을 그리면서 회전하고 다른 공은 y축을 중심으로 타원을 움직이는 것을 구현해보자.
3. Lab06-4을 이용하여 공이 땅에 떨어지면 다시 위로 올라가는 것을 구현해보자.

- 최종제출할 프로그램은 각각의 프로그램을 개선한것이다. 즉 lab06-1-m, lab06-2-m, lab06-4-m 프로그램을 압축하여 제출하면 된다.

실습문제 제출

이번 실습문제는

- 모든 프로그램은 압출하여 하나의 파일로
- PLMS Lab xx 해당 번호폴더
- 마감은 11/30

제출합니다.