1월 3주차 레포트

A. Frame 수정

[해결]

(1). Unit Box를 그리는데 계산량이 많다.

```
def createVertexArraySeparate(self):
        varr = np.array([
            (0,0,1),         # v0 normal
            ( -1 ,  1 ,  1 ), # v0 position
            (0,0,1),         # v2 normal
            (  1 , -1 ,  1 ), # v2 position
            (0,0,1),         # v1 normal
            (  1 ,  1 ,  1 ), # v1 position

            (0,0,1),         # v0 normal
            ( -1 ,  1 ,  1 ), # v0 position
            (0,0,1),         # v3 normal
            ( -1 , -1 ,  1 ), # v3 position
            (0,0,1),         # v2 normal
            (  1 , -1 ,  1 ), # v2 position

            (0,0,-1),
            ( -1 ,  1 , -1 ), # v4
            (0,0,-1),
            (  1 ,  1 , -1 ), # v5
            (0,0,-1),
            (  1 , -1 , -1 ), # v6

            (0,0,-1),
            ( -1 ,  1 , -1 ), # v4
            (0,0,-1),
            (  1 , -1 , -1 ), # v6
            (0,0,-1),
            ( -1 , -1 , -1 ), # v7

            (0,1,0),
            ( -1 ,  1 ,  1 ), # v0
            (0,1,0),
            (  1 ,  1 ,  1 ), # v1
            (0,1,0),
            (  1 ,  1 , -1 ), # v5

            (0,1,0),
            ( -1 ,  1 ,  1 ), # v0
            (0,1,0),
            (  1 ,  1 , -1 ), # v5
            (0,1,0),
            ( -1 ,  1 , -1 ), # v4

            (0,-1,0),
            ( -1 , -1 ,  1 ), # v3
            (0,-1,0),
            (  1 , -1 , -1 ), # v6
            (0,-1,0),
            (  1 , -1 ,  1 ), # v2

            (0,-1,0),
            ( -1 , -1 ,  1 ), # v3
            (0,-1,0),
            ( -1 , -1 , -1 ), # v7
            (0,-1,0),
            (  1 , -1 , -1 ), # v6

            (1,0,0),
            (  1 ,  1 ,  1 ), # v1
            (1,0,0),
            (  1 , -1 ,  1 ), # v2
            (1,0,0),
            (  1 , -1 , -1 ), # v6

            (1,0,0),
            (  1 ,  1 ,  1 ), # v1
```

```
            (1,0,0),
            (  1 , -1 , -1 ), # v6
            (1,0,0),
            (  1 ,  1 , -1 ), # v5

            (-1,0,0),
            ( -1 ,  1 ,  1 ), # v0
            (-1,0,0),
            ( -1 , -1 , -1 ), # v7
            (-1,0,0),
            ( -1 , -1 ,  1 ), # v3

            (-1,0,0),
            ( -1 ,  1 ,  1 ), # v0
            (-1,0,0),
            ( -1 ,  1 , -1 ), # v4
            (-1,0,0),
            ( -1 , -1 , -1 ), # v7
            ], 'float32')
        return varr

def drawCube_glDrawArray(self):
        glColor3f(0.5,0.5,0)
        varr = self.createVertexArraySeparate()
        glEnableClientState(GL_VERTEX_ARRAY)
        glEnableClientState(GL_NORMAL_ARRAY)
        glNormalPointer(GL_FLOAT, 6*varr.itemsize, varr)
        glVertexPointer(3, GL_FLOAT, 6*varr.itemsize, ctypes.c_void_p(varr.ctypes.data + 3*varr.itemsize))
        glDrawArrays(GL_TRIANGLES, 0, int(varr.size/6))

>> 이처럼 UnitBox를 나타내는 배열(삼각형 점들과 그 삼각형에 대한 Normal Vector)
>> UnitBox 한면씩 그리는 것이 아니라, 전체의 정보를 가지고 한꺼번에 그리는 형식
>> 출처 : https://github.com/bh981013/Computer-graphics/blob/268c72e1f6d1ae41aeaef7abaef332d95adf221d/2017029616-7-1.py
>> 출처 : 장보경 학생
```

(2). Walk BVH 파일에서 가져올 때, 처음부터 Rotation Matrix & Vector 계산하기

```
def BVH_read(self):
        f = open("sample-walk.bvh",'r')
        while self.Num < 99:
            lines = f.readline()
            self.Num +=1
        if self.Num >=99:
            while self.Num <298 :
                lines = f.readline()
                lines = lines.split()
                lines = np.array(list(map(float,lines))).T
                self.BVH_pose = np.vstack([self.BVH_pose,lines])
                self.Num +=1
            self.BVH_pose = np.delete(self.BVH_pose,(0),axis=0)
            for i in range(199):
                for j in range(3):
                    self.BVH_pose[i][j] = 0
        f.close()

>> BVH 파일에서 Frame에 따른 Pose부분만 읽어온다


        for i in range(199):
            for j in range(16):
                Z_angle=self.BVH_pose[i][3*j]*math.pi/180
                X_angle=self.BVH_pose[i][3*j+1]*math.pi/180
                Y_angle=self.BVH_pose[i][3*j+2]*math.pi/180

                Z_ROT=np.array([[np.cos(Z_angle),-np.sin(Z_angle),0],
                                [np.sin(Z_angle),np.cos(Z_angle),0],
                                [0,0,1]])
                X_ROT=np.array([[1,0,0],
                                [0,np.cos(X_angle),-np.sin(X_angle)],
                                [0,np.sin(X_angle),np.cos(X_angle)]])
                Y_ROT=np.array([[np.cos(Y_angle),0,np.sin(Y_angle)],
                                [0,1,0],
                                [-np.sin(Y_angle),0,np.cos(Y_angle)]])
                ROT = Z_ROT@ X_ROT @ Y_ROT
                VEC = R.from_matrix(ROT).as_rotvec()
```

```
                for k in range(3):
                    self.BVH_VEC[i][3*j+k] = VEC[k]

>> 읽어온 Pose 정보들을 가지고 Rotation Matrix를 만든다
>> 그 후에 Rotation Matrix를 가지고 Rotation Vector를 만들어서 저장하고 있는다.

$$ 의문점
>> Rotation Matrix의 경우에는 Rotation Vector와 달리 어떻게 저장하는 것이 효율적일까?
>> Rotation Matrix도 Joint와 Frame에 따라 저장되는 형식으로 하여서
>> 예를 들어, Joint : 16, Frame : 200, 일시
>> Rotation Matrix의 형식은 numpy에서 [200][16][4][4] 이런 형식의 고차원 배열인가??
```

(3). Frame 잘못 설정

```
def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glPushMatrix()
        self.world.step() >> DELETE
        self.Camera()
        #self.Skel_Camera()
        self.Drawbaseline()
        self.Drawskeleton()
        glPopMatrix()

        glFlush()

        for i in range(30):
            self.world.step()
            #self.PD_control()
            self.SPD_control()

        if self.frame >= self.end:
            self.frame =self.start
        elif self.frame < self.end :
            self.frame+=1

>> 코드를 자세히 살펴보니, PyQt5의 timer가 돌아갈 때마다 자동으로 업데이트되는
>> paintGL에서 world.step()이 있어서 이것 또한 frame 상에서 문제를 일으켰을 것으로 보인다.
>> 지금 현재는 코드상에서 지웠다.

>> 지금 현재 Frame 상태
>> PyQt5 Timer Frequency : 30 Hz
>> Simulation & Frame Update Frequency : 30 Hz
>> Simulation Time step (Frequency) : 0.00111...s ( 900 Hz)
```

B. SPD Control & PD Control

(1). PD Control

```
def PD_control(self):
        Kp = 500
        Kd = 0.5

        position = self.Human.getPositions()
        velocity = self.Human.getVelocities()

        >> 각각의 Joint마다 정보를 받아오는 것이 아닌 한꺼번에 받아오는 형식으로 변경

        LOG = np.zeros(48)

        for i in range(16):
            pos = np.array([position[3*i],position[3*i+1],position[3*i+2]])
            vel = np.array([velocity[3*i],velocity[3*i+1],velocity[3*i+2]])

            ROTR = (R.from_rotvec(pos)).as_matrix()
            D_VEC = np.array([self.BVH_VEC[self.frame][3*i],self.BVH_VEC[self.frame][3*i+1],self.BVH_VEC[self.frame][3*i+2]])
            ROTD=(R.from_rotvec(D_VEC)).as_matrix()
            LOG_VEC=(R.from_matrix(ROTR.T@ROTD)).as_rotvec()
```

```
            for k in range(3):
                LOG[3*i+k] = LOG_VEC[k]

        >> Loganithm을 통하여 Rotation Vector를 구하기 위해
        >> 전체의 Joint에 대한 정보를 지닌 배열을 3개씩(Z,X,Y)순으로 짤라서 사용

        Torque=Kp*LOG+Kd*(0-velocity)

        for i in range(6):
            Torque[i] = 0

        self.Human.setForces(Torque)

        >> Torque를 줄 때에도 한꺼번에 주는 형식으로 변경
```

(2). SPD Control

```
def SPD_control(self):
        Kp=1000
        Kd=0.4

        position = self.Human.getPositions()
        velocity = self.Human.getVelocities()

        invM = np.linalg.inv(self.Human.getMassMatrix() + Kd * self.world.getTimeStep())
        PP = -Kp * (position + velocity * self.world.getTimeStep() - self.BVH_VEC[self.frame])
        DD = -Kd * velocity
        QDDOT = invM @ (-self.Human.getCoriolisAndGravityForces() + PP + DD + self.Human.getConstraintForces())
        Torque = PP + DD - Kd * QDDOT * self.world.getTimeStep()

        for i in range(6):
            Torque[i] = 0

        self.Human.setForces(Torque)
```

(3). 의문점

PD Control의 Gain(Kp, Kd)이 왜이리 큰 것인가?

>> 장보경 학생의 PD Control

Kp : 1 / Kd : 0.05 / Simulation Time Step : 0.001

Inertia Moment(Heap) : 0.00833

>> 내 PD Control

Kp : 500 / Kd : 0.5 / Simulation Time Step : 0.001

Inertia Moment(Heap) : 0.1847

>>> 아무래도 내가 만든 Skeleton 파일의 Body Node들의 관성모멘트 자체가 커서

>>> 사용된 PD Control Gain(Kp, Kd)가 커진 것이 아닐까 라는 생각이 든다

>>> Heap만 보았을 시, 관성모멘트는 Ixx : 22배, Iyy : 37배, Izz : 50배 정도로

>>> 평균 36배 정도 차이나는 것으로 보인다.

>>> 그렇다면, Gain이 커진다는 것은 이해가 되지만,

>>> 왜 500배나 될 정도로 엄청 커지는 것일까?

Name	장보경 학생	내꺼	
엉덩이			36.3
Ixx	0.00833	0.1874	22.5
Iyy	0.00833	0.304	36.5
Izz	0.00833	0.4165	50.0
가슴			31.8
Ixx	0.00666	0.1311	19.7
Iyy	0.00666	0.2128	32.0
Izz	0.00666	0.2916	43.8
얼굴			18.9
Ixx	0.00125	0.02602	20.8
Iyy	0.00125	0.01875	15.0
Izz	0.00125	0.02602	20.8
왼쪽 위팔			5.2
Ixx	0.00083	0.003333	4.0
Iyy	0.00208	0.01208	5.8
Izz	0.00208	0.01208	5.8
왼쪽 아래팔			9.7
Ixx	0.00029	0.0016	5.5
Iyy	0.00073	0.008611	11.8
Izz	0.00073	0.008611	11.8
왼손			14.8
Ixx	5E-05	0.000625	12.5
Iyy	9E-05	0.001625	18.1
Izz	9E-05	0.00125	13.9

Name	장보경 학생	내꺼	
오른쪽 위팔			5.2
Ixx	0.00083	0.003333	4.0
Iyy	0.00208	0.01208	5.8
Izz	0.00208	0.01208	5.8
오른쪽 아래팔			9.7
Ixx	0.00029	0.0016	5.5
Iyy	0.00073	0.008611	11.8
Izz	0.00073	0.008611	11.8
오른손			14.8
Ixx	5E-05	0.000625	12.5
Iyy	9E-05	0.001625	18.1
Izz	9E-05	0.00125	13.9
왼쪽 허벅지			4.9
Ixx	0.01458	0.06904	4.7
Iyy	0.00292	0.01562	5.3
Izz	0.01458	0.06904	4.7
왼쪽 종아리			19.5
Ixx	0.00281	0.0708	25.2
Iyy	0.00083	0.006666	8.0
Izz	0.00281	0.0708	25.2
왼발			7.3
Ixx	0.00063	0.005167	8.2
Iyy	0.00063	0.005437	8.6
Izz	0.00025	0.00123	4.9
오른쪽 허벅지			4.9
Ixx	0.01458	0.06904	4.7
Iyy	0.00292	0.01562	5.3
Izz	0.01458	0.06904	4.7
오른쪽 종아리			19.5
Ixx	0.00281	0.0708	25.2
Iyy	0.00083	0.006666	8.0
Izz	0.00281	0.0708	25.2
오른발			7.3
Ixx	0.00063	0.005167	8.2
Iyy	0.00063	0.005437	8.6
Izz	0.00025	0.00123	4.9

SPD Control의 Gain(Kp, Kd)는 비교적 덜 큰 것인가?

>> 내 SPD Control

Kp : 1000 / Kd : 0.4 / Simulation Time Step : 0.001

>> PD Control의 특징과 SPD Control의 특징이 다른 부분이 있기에,

>> 관성모멘트가 커져도 PD Control의 Gain이 커지는 비율보다는 적게

>> SPD Control의 Gain이 커지는 것인가?

왜 Simulation 캐릭터는 위로 실금실금 올라가는가?

>> 캐릭터가 위로 조금씩 올라간다

>> Root에 힘을 주었는가? (Heap에 Torque를 주었는가?)

>>> 주지않았다

>> 그렇다면, BVH 파일에서 파일을 잘못 읽어온 것이 아닌가?

>>> 제대로 읽었다

>>> Rotation Matrix와 Vector까지 만들어서 제대로 SPD 코드상에 넣었다.

>> 최종적으론, 원래 캐릭터가 올라가는 움직임인 것인가?

>>> 장보경 학생의 Skel 파일로 Simulation을 구현했을 시, 위로 올라가지 않는다.

>>> 내가 만든 Skel 파일에 문제가 있는 것으로 보인다

>>> Skel 파일의 Body의 관성모멘트가 너무 커서 발생한 문제인가?

>>> 내가 만든 Skel 파일은 몸체부분과 팔다리 부분의 질량 & 관성모멘트의 차이가

>>> 장보경 학생이 만든 Skel 파일보다 크다

>>> 내 것과 장보경 학생의 Skel 파일의 팔다리 부분의 질량 & 관성모멘트의 차이는 적다

>>> 이것은 마치 질량이 서로 다른 돌멩이를

>>> 동일한 줄로 묶어서

>>> 같은 속도로 동일한 원주를 회전시켜서

>>> 날렸을 때, 어느쪽이 더 멀리 날아가는지와 같은 문제이지 않을까?

>>> 무거운 돌멩이가 더 멀리 날아간다. (몸체부가 무거운 내 캐릭터가 위로 떠오른다)

>>> Skel 파일의 Joint의 위치가 잘못되어서 발생한 문제인가?

>>> Joint의 위치로 인한 문제는 아닌 것으로 보인다.

>>> 무중력 상태에서 걷기 운동을 할 때, 제자리에서만 걷는 것이

>>> 불가능하지 않을까?

>>> 완전히 회전에너지와 운동에너지를 딱 맞춰서 frame이 돌아가는 것이

>>> 아니라면, 제자리에서 걷는 운동을 하는 것이 불가능한 것이 아닌가?

>>> Skel 파일을 자세히 살펴보니, 캐릭터가 움직이면 지면과 닿을 수 있는 거리였다.

>>> 즉, 발로 땅을 차서 하늘로 뜨는 것이었다.

```
<skeleton name="ground skeleton">
  <mobile>false</mobile>
  <body name="ground">
    <transformation>0 -2 0 0 0 0</transformation>

>> 다리의 길이는 원점으로부터 약 -1.09m 정도였고,
>> 수정하기 전의 지면의 위치는 원점으로부터 약 -1.1m 정도였다.
>> 원점으로부터 -2.0m로 수정하니 약간의 회전은 있지만, 예전처럼 위로 솟아오르진 않는다.
  >> 개별미팅 때 교수님께서 지면하고 닿는 것이 아니냐고 말씀하셨을 때
  >> 조금 더 자세히 살펴봤어야 했다.
```