Computer Graphics

Project #1

20170807 박민기

Indices

- 모델 생성 및 동작 처리

- 행렬 계산 식

- 결과

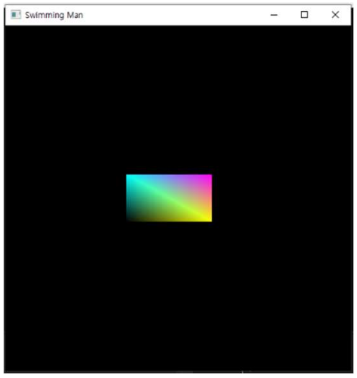
1. 모델 생성 및 동작 처리

* Swimming man 컴퍼넌트(body, arm, leg,...)들의 계층 구조를 조직하기 위해, 가장 최상위 오브젝트인 Body 를 기준으로 설계를 시작하였다. 적당한 크기와 기준인 원점을(0,0,0) 위치로 삼았다.

- **Body**

Scale ==> glm::vec3(1.8, 1, 0.6)

Position ==> glm::vec3(0, 0, 0)

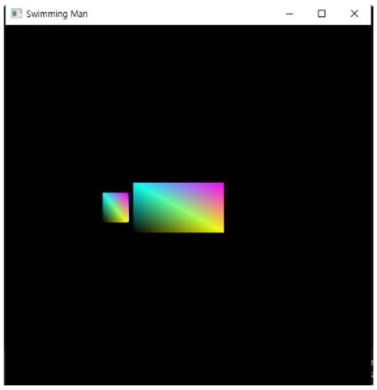


* 이후로는 생성된 body를 기준으로 하여 position 값을 조정함으로써, 상위 오브젝트인 body가 이동하면 자식 객체들도 함께 움직일 수 있도록 계층 구조를 만들어 주었다.

- **Head**

Scale ==> glm:: vec3(0.5, 0.6, 0.2)

Position ==> glm::vec3(bodyPos.x - bodyScale.x + headScale.x, bodyPos.y, bodyPos.z)

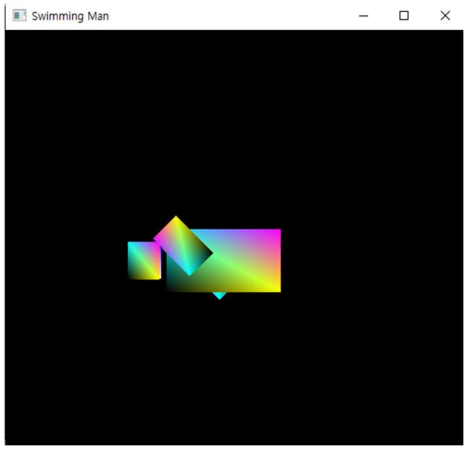


- **Arm**

Scale ==> glm:: vec3(0.8, 0.5, 0.1)

Position ==> glm::vec3(bodyPos.x - (armScale.x / 2), bodyPos.y, bodyPos.z ± (bodyScale.z / 2) ± gap)

// 좌우 팔의 방향의 따라 몸의 양옆으로 붙이기 위해 부호의 방향을 반대로 표시



*** 팔과 다르게 다리는 진자 운동과 비슷하게 최대와 최소를 가지고 범위 내에서만 회전하기 때문에 매 시간마다 각을 비교해서 범위를 벗어나면 상태를 바꿔주는(legUp→legDown) 과정을 별도로 추가하였다.**

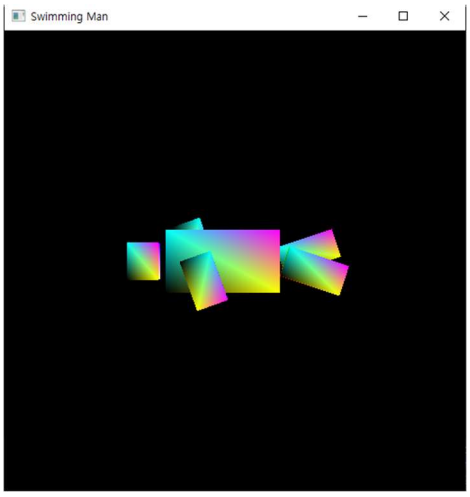
```
if (legRotAngle >= legMaxAngle)
  isLegUp = false;
else if (legRotAngle <= legMinAngle)
  isLegUp = true;

if(isLegUp)
  legRotAngle += glm::radians(t * 360.0f / 5000.0f);
else
  legRotAngle -= glm::radians(t * 360.0f / 5000.0f);
```

- **Leg**

Scale ==> glm:: vec3(1, 0.5, 0.1)

Position ==> glm::vec3(bodyPos.x + upperlegScale.x, bodyPos.y, bodyPos.z ± upperlegScale.z ± gap)

// 좌우 다리의 방향의 따라 몸의 양옆으로 붙이기 위해 부호의 방향을 반대로 표시



*** 위에서 까지의 자식 컴퍼넌트(머리, 팔 다리)는 모두 부모가 body이기 때문에 body의 속성 값에 맞게 position을 조정해주었다. 하지만 남은 아랫팔과 아랫다리의 경우, 그 부모가 각각 body의 자식인 팔과 다리가 된다. 즉, body가 직접 부모가 아닌, 이중으로 연결된 계층 구조를 구성하게 된다.**

이를 위한 처리를 위해서는, 팔과 다리에 했던 처리와 마찬가지로, 이들의 부모를 각각 body가 아닌 팔과 다리로 설정해준다.

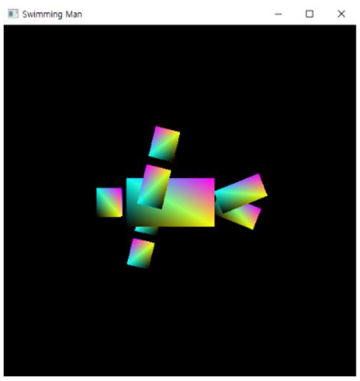
- **Forearm**

Scale ==> glm:: vec3(0.6, 0.5, 0.1)

Position ==> glm::vec3(rightArmPos.x, rightArmPos.y, rightArmPos.z)

// 왼쪽의 경우, 마찬가지로 leftArm의 position값을 기준으로 설정

기준으로 설정해둔 rightArm, 혹은 leftArm이 먼저 body의 자식으로 있기 때문에 아랫팔은 자동적으로 body를 부모로 가지게 된다.

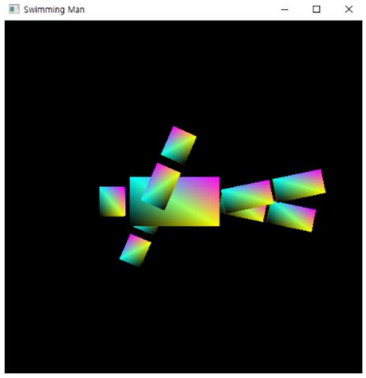


- **Lower leg**

Scale ==> glm:: vec3(1, 0.5, 0.1)

Position ==> glm::vec3(rightUpperlegPos.x, rightUpperlegPos.y, rightUpperlegPos.z)

// 왼쪽의 경우, 마찬가지로 leftLeg의 position값을 기준으로 설정



2. 행렬 계산 식

- **Body**

  기준이 되는 컴퍼넌트이기 때문에 별도의 rotation이 없다. 적절한 scaling 후, view mat과 projection mat의 연산을 거친다.

```
// P * V * T * S * v

modelMat = glm::translate(basis, bodyPos);
modelMat = glm::scale(modelMat, bodyScale);
pvmMat = projectMat * viewMat * modelMat;
```

- **Head**

  적절한 Scaling 이후 부모인 body에 맞춰 translation 과정을 거친다.

```
// P * V * T * S * v

modelMat = glm::translate(basis, headPos);
modelMat = glm::scale(modelMat, headScale);
pvmMat = projectMat * viewMat * modelMat;
```

- **Right Arm**

  팔의 경우부터는 회전이 추가된다. 나아가 일반적으로 글로벌 좌표계에 의한 회전이 아닌, 로컬 좌표계를 기준으로 회전해야 한다.

  예를 들어, 별도의 작업 없이 glm::rotate() 함수를 실행시킬 경우, 팔의 중심을 기준으로 회전하게 된다. 하지만 이번 프로젝트에서 요구하는 것은, 관절을 중심으로 회전하여 사람의 동작을 구현하는 것이기 때문에, 회전을 시키기 전, 좌표 축을 옮기는 작업이 선행되어야 한다. 마지막으로 변경된 좌표축을 기준으로 회전을 마치면, 저장해둔 원래 위치에 맞게 translation 과정을 거친다.

```
// P * V * T * R * S * T * v

modelMat = glm::translate(basis, rightArmPos);   //이후 body에 맞춰 위치 설정
modelMat = glm::rotate(modelMat, armRotAngle, glm::vec3(0, 0, 1)); //회전
modelMat = glm::scale(modelMat, armScale);                  //스케일링
//로컬 회전축으로 회전시키기 위해 임의의 회전축 방향으로 먼저 이동
modelMat = glm::translate(modelMat, glm::vec3(armScale.x/2, 0, 0));
pvmMat = projectMat * viewMat * modelMat;
```

- **Left Arm**

```
// P * V * T * R * S * T * v
```

```
modelMat = glm::translate(basis, leftArmPos);
modelMat = glm::rotate(modelMat, armRotAngle, glm::vec3(0, 0, 1));
modelMat = glm::scale(modelMat, armScale);
modelMat = glm::translate(modelMat, glm::vec3(-armScale.x / 2, 0, 0));
pvmMat = projectMat * viewMat * modelMat;
```

- **Right Upper Leg**

```
// P * V * T * R * S * T * v
```

```
modelMat = glm::translate(basis, rightUpperlegPos);
modelMat = glm::rotate(modelMat, legRotAngle,glm::vec3(0, 0, 1));
modelMat = glm::scale(modelMat,upperlegScale);
modelMat = glm::translate(modelMat, glm::vec3(upperlegScale.x/2, 0, 0));
pvmMat = projectMat * viewMat * modelMat;
```

- **Left Upper Leg**

다리의 경우, 양쪽이 다른 방향으로 움직여야 하기 때문에 회전각의 부호를 반전시켜 적용한다.

```
// P * V * T * R * S * T * v
```

```
modelMat = glm::translate(basis, leftUpperlegPos);
modelMat = glm::rotate(modelMat, -legRotAngle, glm::vec3(0, 0, 1));
modelMat = glm::scale(modelMat, upperlegScale);
modelMat = glm::translate(modelMat, glm::vec3(upperlegScale.x/2, 0, 0));
pvmMat = projectMat * viewMat * modelMat;
```

- **Right Forearm**

이중 부모를 갖는 아랫팔의 경우, 기준 대상을 윗팔로 설정하여 position값을 설정한다. 다만 이번 문제에서는 아랫팔과 윗팔이 1자를 유지한 형태로 회전해야 하기 때문에 회전의 중심은 윗팔과 같게 설정한다.

```
// P * V * T * R * S * T * v
```

```
modelMat = glm::translate(basis, rightForearmPos);
modelMat = glm::rotate(modelMat, armRotAngle, glm::vec3(0, 0, 1));
modelMat = glm::scale(modelMat, forearmScale);
modelMat = glm::translate(modelMat,
glm::vec3(armScale.x+(forearmScale.x/2)+ armRotGap, 0, 0));
pvmMat = projectMat * viewMat * modelMat;
```

- **Left Forearm**

```
// P * V * T * R * S * T * v
```

```
modelMat = glm::translate(basis, leftForearmPos);
modelMat = glm::rotate(modelMat, armRotAngle, glm::vec3(0, 0, 1));
modelMat = glm::scale(modelMat, forearmScale);
modelMat = glm::translate(modelMat, glm::vec3(-(armScale.x +
(forearmScale.x / 2) + armRotGap), 0, 0));
pvmMat = projectMat * viewMat * modelMat;
```

- **Right Lower Leg**

다리도 마찬가지로, body가 아닌 윗다리를 부모로 설정한다. 또한 윗다리와 맞춰 1자로 회전해야 하기 때문에 그 중심을 윗다리와 동일하게 설정한다. 회전 방향은 좌우 같은 속도, 다른 방향을 구현하기 위해서 한 쪽의 회전각 부호를 반대로 설정하였다.

```
// P * V * T * R * S * T * v
```

```
modelMat = glm::translate(basis, rightLowerlegPos);
modelMat = glm::rotate(modelMat, legRotAngle, glm::vec3(0, 0, 1));
modelMat = glm::scale(modelMat, lowerlegScale);
modelMat = glm::translate(modelMat, glm::vec3(upperlegScale.x +
(lowerlegScale.x / 2) + legRotGap, 0, 0));
pvmMat = projectMat * viewMat * modelMat;
```

- **Left Lower Leg**

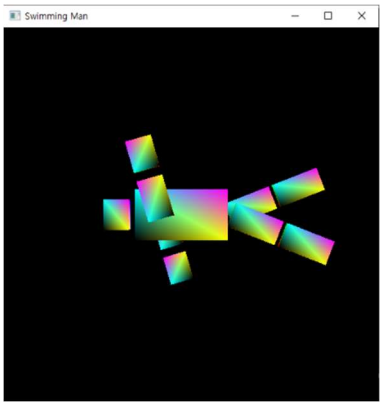
```
// P * V * T * R * S * T * v
```

```
modelMat = glm::translate(basis, leftLowerLegPos);
modelMat = glm::rotate(modelMat, -legRotAngle, glm::vec3(0, 0, 1));
modelMat = glm::scale(modelMat, lowerlegScale);
modelMat = glm::translate(modelMat, glm::vec3(upperlegScale.x +
(lowerlegScale.x / 2) + legRotGap, 0, 0));
pvmMat = projectMat * viewMat * modelMat;
```

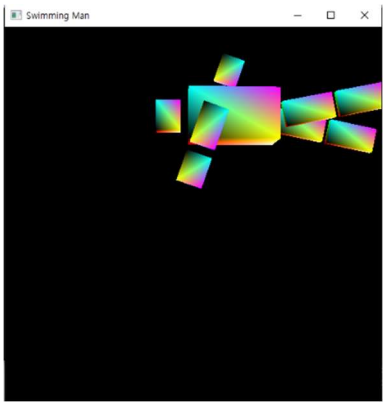
3. 결과

위와 같은 계층 구조를 맞추어 만들어진 이번 모델의 경우, 고정된 몸통과 머리에 회전하는 팔과 다리를 확인할 수 있었다.

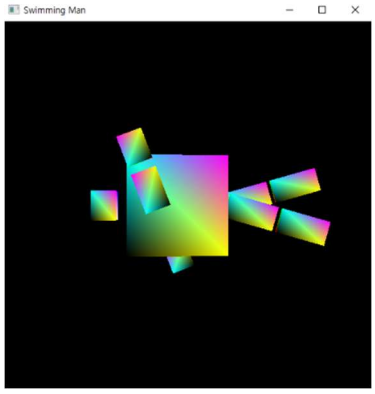
다만 몸통을 중심으로 계층이 존재하기 때문에 몸통의 position을 변경라는 등의 변화를 주는 경우에도 자식 객체들도 함께 이동하는 것 역시 확인할 수 있었다.



*** 기본 모델**



*** Body의 Position 값 변경**



*** Body의 Scale 값 변경**