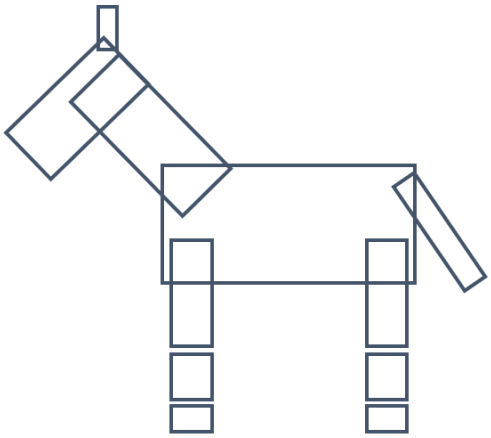
Computer Graphics Project 1 – Cube Animal Walking



제가 구현한 동물은 말입니다. 기본 모델에서 다리에 큐브를 하나 추가하였고, 목 부분과 귀를 추가하였습니다. 말의 움직임의 경우 유튜브 영상[1]을 참고했습니다.

기존 코드에서 변화된 부분은 다음과 같습니다 :

```
worldMat = glm::rotate(glm::mat4(1.0f), rotAngle, glm::vec3(1.0f, 1.0f, 0.0f));
```

1. 기존 코드는 고정된 하나의 벡터를 축으로 회전하기 때문에 모델을 원하는 대로 관찰하기 힘들었습니다. T를 눌렀을 때 회전 축이 x, y, z 축으로 전환되도록 구현했습니다.
2. C를 누르면 자동차가 나왔던 기존의 코드에서 자동차에서 한 번 더 누르면 cube animal walking이 나오도록 구현했습니다.
3. worldMat을 회전하는데 사용되는 rotAngle 변수에 따라 바퀴가 움직였던 기존 코드에서 별개의 변수에 따라 회전하도록 바뀌었습니다.
4. 기본적으로 자동차의 바퀴와 동물이 움직이고 M을 누를 시 멈추도록 구현되었습니다.

Cube Animal을 구현한 방법입니다.

동물의 각 부분에 다음 과정을 반복합니다.

1. 각 부분을 기본 위치로 translate 합니다.

   이때 사용되는 vector는 상수입니다.

[1] https://www.youtube.com/watch?v=Ggq3vlQQp1U&ab_channel=MollyEricson

2. 각 부분을 기본 각도로 rotate 합니다.

   이 때 사용되는 각도와 축은 상수입니다.

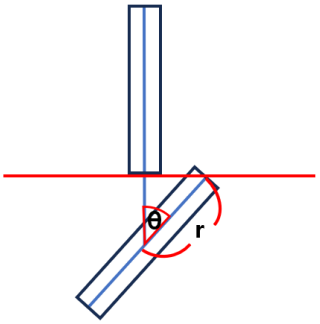
3. 움직이는 부분의 경우 움직이는 부분에 따라 rotate합니다.

   a. 각 관절 angle을 parameter로 rotate 함수를 호출합니다.

      이때 이 관절에 연결된 모든 하위 계층을 같이 회전합니다.

      각 관절의 angle 계산의 경우 조금 뒤에 설명하겠습니다.

   b. rotate하면서 틀어진 위치를 translate 함수로 보정합니다.



   -r * sin(θ) 만큼 translate 해서 rotate한 부분을 다시 관절에 붙여줍니다.

4. 각 부위의 크기를 scale matrix를 이용해 조절합니다.

5. 계산된 각 부위의 modeling matrix, projection matrix, view matrix를 곱해 이를 그립니다.

한 부위를 코드로 구현한 결과는 다음과 같습니다.

1. 초기 위치로 translate

```
//translate each bone to position
backLegMat_1[i] = glm::translate(animalMat, backLegPos[i]);
backLegMat_2[i] = glm::translate(animalMat, backLegPos[i]);
backLegMat_3[i] = glm::translate(animalMat, backLegPos[i]);
backLegMat_2[i] = glm::translate(backLegMat_2[i], glm::vec3(0, 0, -(bLength1 + bLength2) /
2));
backLegMat_3[i] = glm::translate(backLegMat_3[i], glm::vec3(0, 0, -((bLength1 + bLength3) / 2
+ bLength2)));
```

2. 관절에 연결된 하부 부위를 모두 회전

```
backLegMat_1[i] = glm::rotate(backLegMat_1[i], backJointAngle[i], glm::vec3(0, 1, 0));
backLegMat_2[i] = glm::rotate(backLegMat_2[i], backJointAngle[i], glm::vec3(0, 1, 0));
backLegMat_3[i] = glm::rotate(backLegMat_3[i], backJointAngle[i], glm::vec3(0, 1, 0));
```

3. 회전에 따른 위치보정

```
frontLegMat_1[i] = glm::translate(frontLegMat_1[i], glm::vec3(-fLength1 * glm::sin(legAngle)/2
* dir, 0, 0));
frontLegMat_2[i] = glm::translate(frontLegMat_2[i], glm::vec3(-(fLength2/2 + fLength1) *
glm::sin(legAngle) * dir, 0, 0));
```

```
frontLegMat_3[i] = glm::translate(frontLegMat_3[i], glm::vec3(-(fLength3/2 + fLength1 + fLength2)
* glm::sin(legAngle) * dir, 0, 0));
```

4. 계산된 modeling vector를 이용해 그리기

```
pvmMat = projectMat * viewMat * frontLegMat_1[i];
glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
glDrawArrays(GL_TRIANGLES, 0, NumVertices);
```

각 관절의 Angle 계산은 idle function에 구현했습니다.

```
static int moveCycle = 4000;
```

움직임 주기를 4000ms 로 설정했습니다.

```
int cycleTime = 0;
int backCycleTime = 500;
cycleTime += abs(currTime - prevTime);
backCycleTime += abs(currTime - prevTime);
```

cycleTime, backCycleTime은 다리의 움직임 주기 시간을 나타내는 변수입니다. 말은 앞다리 뒷다리가 약간 어긋난 채로 움직이므로 두 변수를 따로 선언했습니다.

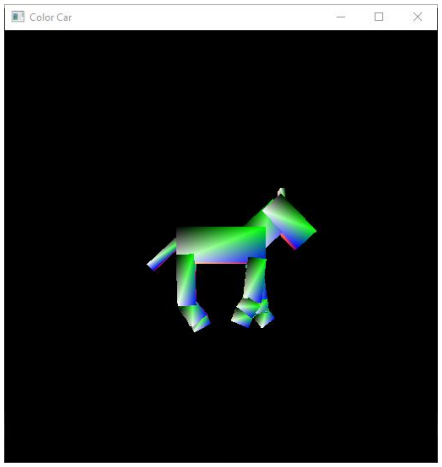
```
int oppositeCycle = (cycleTime + moveCycle / 2) % moveCycle;
int backOppositeCycle = (backCycleTime + moveCycle / 2) % moveCycle;
```

두 다리가 반대로 움직이므로 반대 다리의 시간도 계산합니다.

```
//front leg joint 1
if (cycleTime > 0 && cycleTime <= 1000)
        rontJointAngle1[1] = glm::radians(45.0f - cycleTime * 45.0f / 1000.0f);
else if (cycleTime > 3000 && cycleTime <= 4000)
        frontJointAngle1[1] = glm::radians((cycleTime - 3000) * 45.0f / 1000.0f);
if (oppositeCycle > 0 && oppositeCycle <= 1000)
        frontJointAngle1[0] = glm::radians(45.0f - oppositeCycle * 45.0f / 1000.0f);
else if (oppositeCycle > 3000 && oppositeCycle <= 4000)
        frontJointAngle1[0] = glm::radians((oppositeCycle - 3000) * 45.0f / 1000.0f);
```

각 움직이는 부분의 회전각도를 위의 시간 변수에 따라 변화하도록 설정해줍니다.

시간 변수 값에 따라 **if-else** 문을 단 한 번 만 돌도록 만들 수도 있지만 새로운 움직임을 시도해볼 때 코드 수정의 번거로움 때문에 이렇게 구현했습니다.



실행화면은 다음과 같습니다.

코드를 시험해 보실 때 새로 추가한 버튼 T와 M을 이용하시길 추천드립니다.