컴퓨터그래픽스 과제 2

소프트웨어학과 20003325 이태혁

실행결과

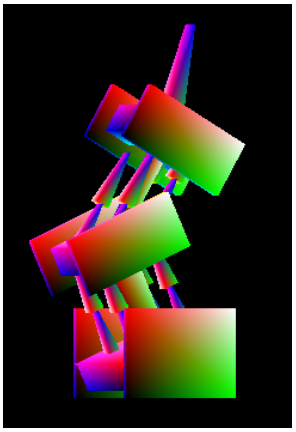
풀이과정

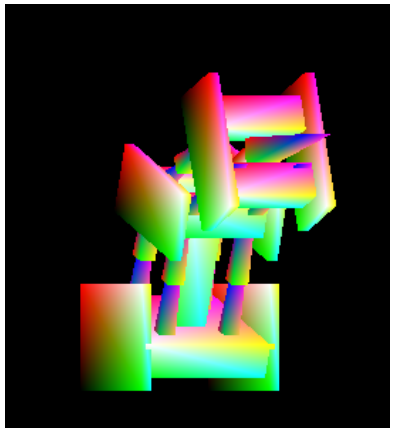
이번 과제는 크게 두 가지 과정으로 나뉜다. RobotArm의 모양을 보기좋게 생성하는 작업과 3개의 각도를 조절하여 미리 주어진 target에 RobotArm의 끝단을 가져다 두는 작업이다.

1. Robot Arm 생성

3개의 관절을 가지고 있는 Robot Arm 을 생성한다. 이 과정을 수행하면서 CTM을 stack구조로 만들어서 사용하는 원리에 대해 완벽하게 이해할 수 있었다.
로봇의 모양을 어떻게 디자인할지 생각하는데 의외로 많은 시간이 걸렸다. 중앙에 main기둥을 세우고 그 주변으로 4개의 작은 뼈대를 심어주기로 했다. 그림을 그려나가다 보니 인간의 척추와 비슷한 모양으로 그려지는 것 같아서 아예 컨셉을 척추 모양으로 잡고 그려나갔다.

- 다양한 각도에서 캡쳐한 Robot Arm



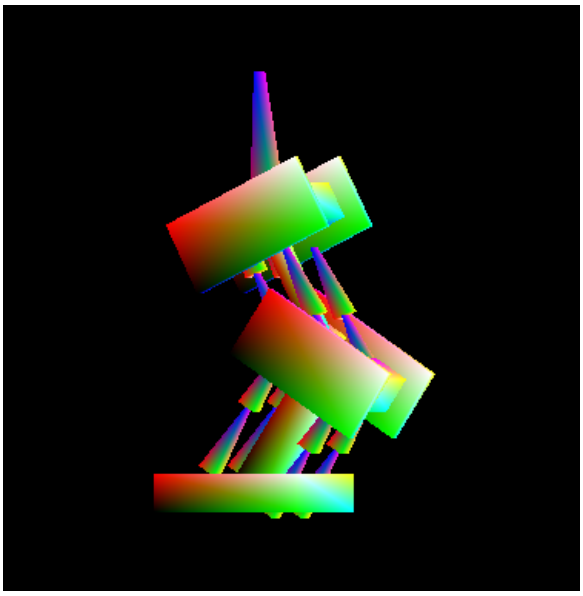


```
CTM = Translate(0, -0.4, 0) * RotateY(g_time * 30);
mat4 temp = CTM; // target의 원점 세팅을 위해 저장
```

가장 처음에 CTM을 y축으로 0.4만큼 내려주어서 Base가 -0.4부터 시작하도록 하였다.
이때의 CTM을 temp에 저장해두어서 나중에 target cube를 그릴 때 이 지점을 축으로 하여 공전하도록 하였다.

각 관절의 중심부를 CMT 스택에 저장하고, 한쪽 뼈대를 쌓아 올린 후 스택구조를 이용하여 다시 관절로 돌아와 다른 뼈대를 쌓아올리는 식으로 구현하였다.

관절을 구현할 때 고려해야할 것들이 몇 가지가 있었다. 처음에는 target이 3차원 상에서 자유롭게 돌아가는 것 처럼 보였는데, y축 회전을 멈추고 보면 사실 target의 Z좌표는 변하지 않고, XY평면 위에서 그저 원 운동을 하고 있다는 것을 알게 되었다. 따라서, 자연스러운 관절의 움직임을 구현하기 위해 Z축 방향으로 양 끝단에 받침대를 놓았고, 그 사이에 있는 XY평면 위에서 뼈대가 회전운동을 하도록 구현하였다.



[그림 1] 부자연스러운 움직임

받침대 사이에서 베이스가 함께 돌아가야 자연스러운 관절의 움직임이 생기는데, [그림 1]과 같이 베이스를 고정시키고 뼈대부터 회전하게 되면 베이스 아래로 뼈가 튀어나오는 부자연스러운 움직임이 생기게 된다. 모든 관절에는 Z축 상에 받침대를 놓아주고, 그 사이에 부품이 끼워진 채 회전운동을 해야 자연스러워 진다. 태블릿pc 거치대 같은 물체를 생각하며 움직임을 구현하는 것이 많은 도움이 되었다.

2. Compute Angle 구현

MyTarget 헤더파일에 들어있는 GetPosition 메소드로 target의 좌표를 받아온다. 그리고 RobotArm의 끝단과의 거리를 계산하여 그 거리가 최소가 되는 순간의 ang1, 2, 3을 구해내고, 그 각들을 RobotArm에 적용하여 그림을 그리는 방식으로 구현하였다.

위에 1번 과정에서 언급했듯이 target의 Z좌표는 0으로 고정되어있기 때문에 X, Y좌표만 고려하여 거리를 계산한다. RobotArm 끝단의 좌표는 각이 3개가 주어지므로, 관절마다 Arm의 길이에 삼각함수를 곱하고, 각각의 Arm에서 나온 X성분과 Y성분을 더해주어서 좌표를 계산하였다.

```
--- RobotArm의 X 좌표---
robot_pos.x = 0.4 * -sin(3.141592 / 180 * i) + 0.4 * -sin(3.141592 / 180 * (i +
 j)) + 0.5 * -sin(3.141592 / 180 * (i + j + k));

--- RobotArm의 Y 좌표---
robot_pos.y = 0.4 * cos(3.141592 / 180 * i) + 0.4 * cos(3.141592 / 180 * (i + j))
 + 0.5 * cos(3.141592 / 180 * (i + j + k));

--- RobotArm과 target 사이의 거리 ---
dist = sqrt((robot_pos.x - target_pos.x) * (robot_pos.x - target_pos.x) + (robot_p
os.y - target_pos.y) * (robot_pos.y - target_pos.y)
```

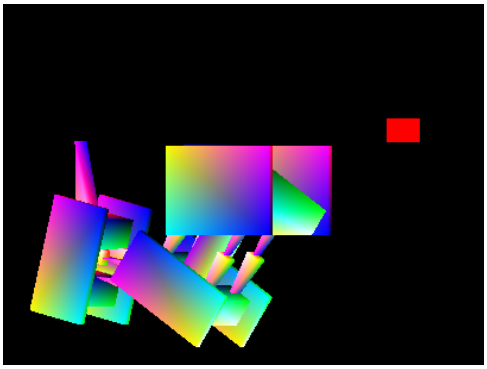
그 후에, 반복문을 돌면서 dist가 0.1 이하인 순간의 각도들을 ang1, 2, 3에 적용하도록 하였는데, 처음에는 0.1이 아닌 어려가지 상수로 시도를 하다가, 0.1정도일 때 target을 가장 잘 추적하는 것으로 보여서 그렇게 설정했다.

```
if (dist < 0.1) // 거리가 0.1 이하이면 각도 저장
      {
        ang1 = i;
        ang2 = j;
        ang3 = k;
        break;
      }
```

▼ 시행착오

1. RobotArm의 좌표 계산 시 삼각함수 사용에 대한 고찰

target 추적 중에 RobotArm이 target과 반대방향을 가리키는 현상이 나타났다.



X좌표가 target의 반대쪽을 가리킴

보통 삼각함수로 좌표를 표현한다고 하면, cos이 X좌표, sin이 Y좌표를 나타낸다. 하지만 RobotArm의 경우 초기상태가 Y축의 양의방향을 향하고 있고, 각도가 90도부터 시작하게 되므로, cos이 Y좌표, sin이 X좌표를 표현하게 된다. 또한, X좌표가 0~180도 까지 음수가 되기 때문에 sin그래프를 고려했을 때 sin에 마이너스를 붙여서 X좌표를 구현해야 정상적인 target 추적이 가능하다.

---

2. 끝단으로 갈 수록 각도를 중첩해야 한다

```
--- 각도를 잘못 설정한 코드 ---
robot_pos.x = 0.4 * -sin(3.141592 / 180 * i) + 0.4 * -sin(3.141592 / 180 * j) +
0.5 * -sin(3.141592 / 180 * k);

--- 올바른 코드 ---
```

```
robot_pos.x = 0.4 * -sin(3.141592 / 180 * i) + 0.4 * -sin(3.141592 / 180 * (i +
j)) + 0.5 * -sin(3.141592 / 180 * (i + j + k));
```

RobotArm의 좌표를 처음 계산할 때 구현한 코드이다. 이렇게 하니 뭔가 방향은 맞는 것 같은데 target의 속도를 못따라가고 놓쳐버리는 현상이 발생했다. 원인은 각도 설정에 있었다. Upper Arm이 회전하면 Rower Arm의 초기값은 Upper Arm에서 부터 시작하므로, Upper Arm에 적용된 각도를 더해주어야 정확하게 target을 따라갈 수 있다. 즉, Arm의 끝단으로 갈 수록 이전 관절에서의 각도를 중첩해주어야 한다는 것이다.

---

3. 실행속도에 따른 고찰

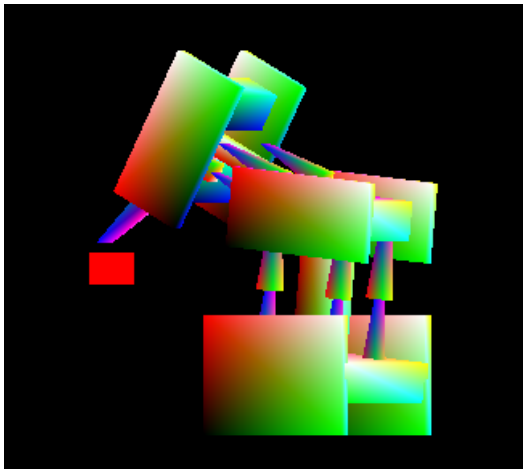
위의 코드들을 전부 구현한 후 실행해보았는데, target을 확실하게 추적하는 건 확인했지만, 속도가 굉장히 느리고 버벅였다. 사실 3중 for문을 사용하여서 속도가 느려질 것이라는 건 어느정도 예상하고 있었다.

일단 for문을 하나 줄이는 것이 우선순위였다. 가장 큰 관절인 ang1은 그대로 두고, ang2와 ang3은 for문 하나로 같이 묶어서, 동시에 증가하도록 구현했다. 또한, 모든 각들을 2씩 증가하도록 하여, target을 조금이나마 빠르게 추적할 수 있도록 하였다.

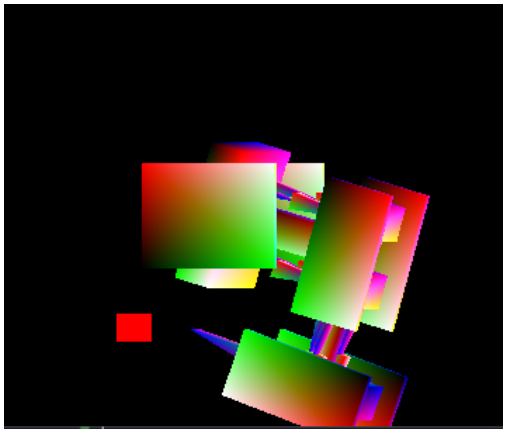
```
--- 실행속도가 느렸 코드 ---

for (int i = 0; i < 360; i++) {
  for (int j = 0; j < 360; j++) {
    for (int k = 0; k < 360; k++) {

      robot_pos.x = ...
      robot_pos.y = ...

--- 개선한 코드 ---

for (int i = 0; i < 360; i += 2) {
    for (int j = 0, k = 0; j < 360; j += 2, k += 2) {

      ...
```

4. Arm의 부자연스러운 움직임

RobotArm이 특정 지점에서 너무 급격하게 정반대 방향으로 꺾이는 현상이 발생했다.



[그림 2]



[그림 3]

```
for (int i = 0; i < 360; i += 2) {
    for (int j = 0, k = 0; j < 120; j += 2, k += 2) {
// ang2, ang3을 360도 --> 120도로 하향 조정함
```

그림 2처럼 왼쪽으로 숙이면서 돌다가 갑자기 그림 3처럼 한번에 반대로 꺾여서 돌아가는 것이다. 이는 각 관절의 각을 360도까지 돌도록 하여서 나타나는 현상으로 판단했다. 사실 각이 3개나 존재하기 때문에 모든 관절을 360로 돌리지 않아도 충분히 target을 추적할 수 있다. 그래서 가장 큰 관절인 ang1은 360도 회전이 가능하도록 하고, ang2와 ang3은 120도 정도만 회전하도록 하여 target을 추적할 때 한쪽 방향으로 자연스럽게 따라갈 수 있도록 하였다.