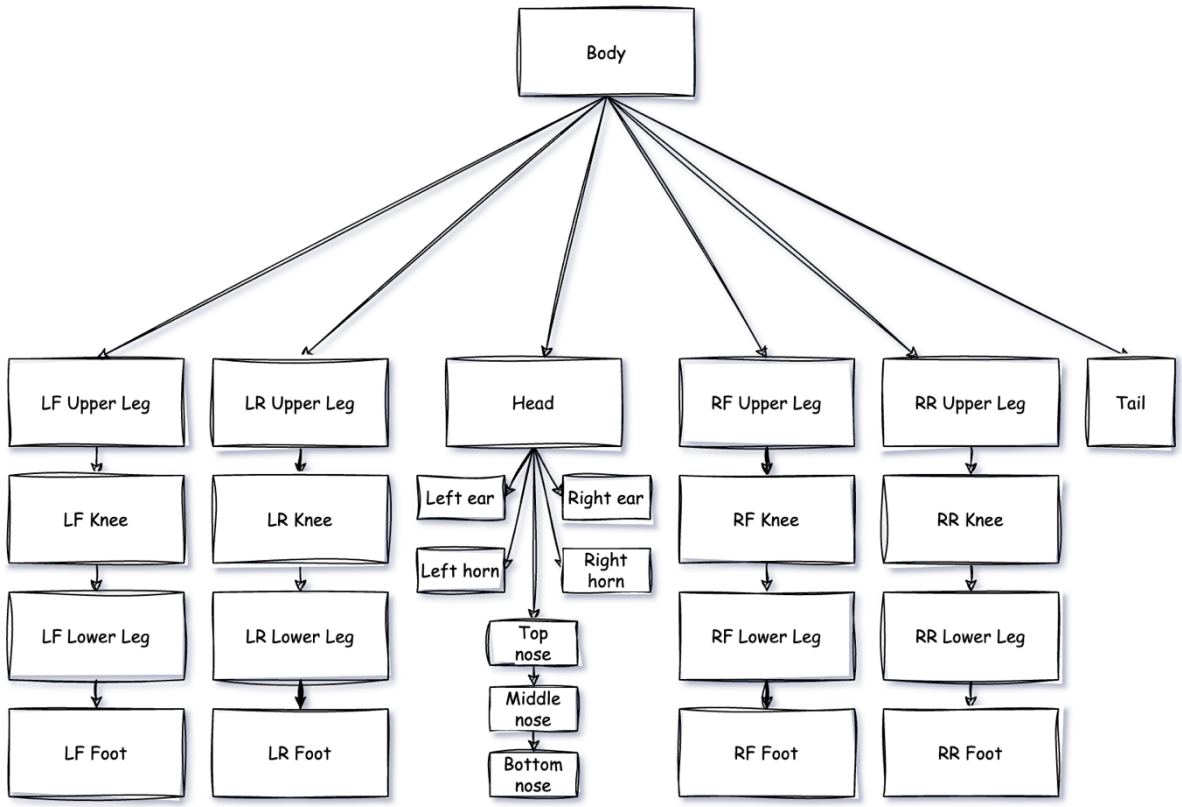
Computer Graphics Project 01

20203361 장민석[1]

[1] 표지의 이미지는 OpenAI, Dall-E 3에 의해 생성된 이미지임

1. 모델 구성

큐브를 이용하여 표현하기로 선택한 모델은 코끼리이다. 코끼리는 상대적으로 다른 동물들에 비해 덩어리감이 있고, 특징이 강한 동물이기 때문에 단순한 도형으로도 효과적으로 표현이 가능할 것으로 기대하였다. 다음의 그래프는 코끼리 모델의 계층 구조이다.



각 노드는 코끼리의 특정 신체 부위를 나타내며, 화살표는 한 부위가 다른 부위에 종속되는 관계를 표시한. 이러한 계층 구조의 부모-자식 관계를 통해 복잡한 움직임을 구현할 수 있다.

Body는 코끼리의 몸통으로, 전체 신체를 대표하는 가장 상위 계층이다. 모든 다른 부위는 이 Body에 연결된다.

코끼리 모델에는 총 4개의 다리가 존재하며, 각 다리는 허벅지(Upper Leg) -> 무릎(Knee) -> 종아리(Lower Leg) -> 발(Foot)의 순으로 연결된다. LF는 왼쪽 앞다리에 해당하며, 마찬가지로, LR는 왼쪽 뒷다리에, RF는 오른쪽 앞다리에, RR은 오른쪽 뒷다리에 해당한다.

Head는 코끼리의 머리 부분으로, 여러 하위 요소를 포함한다. Ears는 두 귀로, 각각 Left ear과 Right ear로 나뉜다. Horns은 두 뿔로 Left horn과 Right horn로 나뉜다. Nose는 코로, Top nose, Middle nose, Bottom nose의 세 부분으로 나뉘어 구성된다.

마지막으로 Tail은 코끼리의 꼬리 부분으로, Body와 연결된다.

2. 모델 생성 및 애니메이션

모델을 그리기에 앞서, 모델을 쉽게 관찰하기 위한 사용자의 keyboard-input에 따른 회전 기능을 구현하였다. 사용자가 '1', '2', '3' key를 누르면 각 축의 각을 저장하는 전역변수의 값이 수정되고, 해당 회전을 위한 worldRotMat 매트릭스를 구성한다. 코드는 다음과 같다.

```
void keyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
    case '1':
        rotAngleWorldx += 0.125f;
        break;
    case '2':
        rotAngleWorldy += 0.125f;
        break;
    case '3':
        rotAngleWorldz += 0.125f;
        break;
    case 033: // Escape key
    case 'q':
    case 'Q':
        exit(EXIT_SUCCESS);
        break;
    }
}

void display(void)
{
    glm::mat4 worldRotMat, pvmMat;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    worldRotMat = glm::rotate(glm::mat4(1.0f), rotAngleWorldx, glm::vec3(1, 0, 0));
    worldRotMat *= glm::rotate(glm::mat4(1.0f), rotAngleWorldy, glm::vec3(0, 1, 0));
    worldRotMat *= glm::rotate(glm::mat4(1.0f), rotAngleWorldz, glm::vec3(0, 0, 1));

    drawElephant(worldRotMat);

    glutSwapBuffers();
}
```

이렇게 구성된 worlRoatMat 매트릭스는 drawElephant 함수의 인자로 전달되어, 모든 요소들을 그릴 때, 사용된다. 다음은 drawElephant 함수이다.

```
void drawElephant(glm::mat4 worldRotMat)
{
    glm::mat4 bodyMat = glm::translate(glm::mat4(1.0f), glm::vec3(0, 0, 0));
    bodyMat = glm::rotate(bodyMat, -rotAngleLeg * 10.0f, glm::vec3(1, 0, 0));

    drawBody(worldRotMat, bodyMat); // 몸통 그리기
    drawHead(worldRotMat, bodyMat); // 머리 그리기
    drawLeg(worldRotMat, bodyMat);  // 다리 그리기
}
```

위의 함수에서는 우선 bodyMat을 구성한다. 이는 코끼리의 body의 tranform에 대한 매트릭스로 body에 종속된 모든 요소들은 해당 매트릭스를 인자로 받고, transform에 사용하여, body의 움직임의 효과를 동일하게 받게 된다. bodyMat에 회전 transform을 적용하여 코끼리가 걸을 때, 좌우로 뒤뚱거리는 모습을 표현하였다. 함수의 내부에서 각각 body를 그리는 함수, head를 그리는 함수, leg를 그리는 함수를 호출한다.

아래는 몸통을 그리는 코드이다.

```
void drawBody(glm::mat4 worldRotMat, glm::mat4 bodyMat)
{
    glm::mat4 modelMat, pvmMat, scaleMat, identityMat = glm::mat4(1.0f);

    // 몸통
    scaleMat = glm::scale(identityMat, glm::vec3(1.4, 1, 0.9));
    pvmMat = projectMat * viewMat * worldRotMat * bodyMat * scaleMat;
    glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
    glDrawArrays(GL_TRIANGLES, 0, NumVertices);

    // 꼬리
    modelMat = glm::translate(identityMat, glm::vec3(-0.8, 0, 0));
    modelMat = glm::rotate(modelMat, 0.35f, glm::vec3(0, 1, 0));
    scaleMat = glm::scale(identityMat, glm::vec3(0.1, 0.1, 0.75));
    pvmMat = projectMat * viewMat * worldRotMat * bodyMat * modelMat * scaleMat;
    glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
    glDrawArrays(GL_TRIANGLES, 0, NumVertices);
}
```

몸통의 경우 로컬 상에서 (0,0,0)에 위치하고, 크기만 tranform해주면 된다. 따라서, pvmMat = projectMat * viewMat * worldRotMat * bodyMat * scaleMat;의 일련의 행렬곱을 수행한다. 차례로, 크기를 조정하고, 인자로 받은 bodyMat에 맞추어 tranlate와 rotate를 진행하고, worldRotMat에 따라 회전한 뒤, viewMat에 맞추어 투영되고, 최종적으로 projectMat에 맞추어 2차원 화면의 좌표로 변환된다. 꼬리는 우선 modelMat에 꼬리 모양을 모두 정의한 뒤, 크기를 조정하는 scaleMat을 구성하거, 마찬가지로 매트릭스들을 차례로 곱해나간다.

아래는 머리를 그리는 코드이다.

```
void drawHead(glm::mat4 worldRotMat, glm::mat4 bodyMat)
{
    glm::mat4 modelMat, pvmMat, scaleMat, identityMat = glm::mat4(1.0f);
    glm::mat4 headMat, noseMat;

    headMat = glm::translate(glm::mat4(1.0f), glm::vec3(0.25f, .0f, -0.2f));
    headMat = glm::rotate(headMat, rotAngleLeg * 35.0f, glm::vec3(1.0f, 0.0f, 0.0f));

    // 머리
    modelMat = glm::translate(identityMat, glm::vec3(0.75, 0, 0.45));
    modelMat = glm::rotate(modelMat, -0.25f, glm::vec3(0, 1, 0));
    scaleMat = glm::scale(identityMat, glm::vec3(0.65, 0.6, 0.65));
    pvmMat = projectMat * viewMat * worldRotMat * bodyMat * headMat * modelMat *
scaleMat;
    glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
    glDrawArrays(GL_TRIANGLES, 0, NumVertices);
```

```
    // 귀
    for (int i = 0; i < 2; i++)
    {
        int sign = i == 0 ? 1 : -1;
        modelMat = glm::translate(identityMat, glm::vec3(0.7, 0.5 * sign, 0.45));
        modelMat = glm::rotate(modelMat, -0.25f, glm::vec3(1 * sign, 1, -1 * sign));
        scaleMat = glm::scale(identityMat, glm::vec3(0.125, 0.65, 0.65));
        pvmMat = projectMat * viewMat * worldRotMat * bodyMat * headMat * modelMat *
scaleMat;
        glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
        glDrawArrays(GL_TRIANGLES, 0, NumVertices);
    }

    // 상아
    for (int i = 0; i < 2; i++)
    {
        int sign = i == 0 ? 1 : -1;
        modelMat = glm::translate(identityMat, glm::vec3(0.8, 0.275 * sign, 0.0));
        modelMat = glm::rotate(modelMat, -.35f, glm::vec3(-1 * sign, 1, -1 * sign));
        scaleMat = glm::scale(identityMat, glm::vec3(0.1, 0.1, 0.65));
        pvmMat = projectMat * viewMat * worldRotMat * bodyMat * headMat * modelMat *
scaleMat;
        glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
        glDrawArrays(GL_TRIANGLES, 0, NumVertices);
    }

    // 코 1
    noseMat = glm::translate(identityMat, glm::vec3(0.85, 0, 0.0));
    // noseMat = glm::rotate(noseMat, 0, glm::vec3(0, 1, 0));
    noseMat = glm::rotate(noseMat, -rotAngleLeg * 35.0f, glm::vec3(0, 0, 1));
    scaleMat = glm::scale(identityMat, glm::vec3(0.45, 0.45, 0.65));
    pvmMat = projectMat * viewMat * worldRotMat * bodyMat * headMat * noseMat *
scaleMat;
    glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
    glDrawArrays(GL_TRIANGLES, 0, NumVertices);

    // 코 2
    noseMat = glm::translate(noseMat, glm::vec3(0, 0, -0.5));
    noseMat = glm::rotate(noseMat, 0.1f, glm::vec3(0, 1, 0));
    noseMat = glm::rotate(noseMat, -rotAngleLeg * 35.0f, glm::vec3(0, 0, 1));
    scaleMat = glm::scale(identityMat, glm::vec3(0.35, 0.35, 0.45));
    pvmMat = projectMat * viewMat * worldRotMat * bodyMat * headMat * noseMat *
scaleMat;
    glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
    glDrawArrays(GL_TRIANGLES, 0, NumVertices);

    // 코 3
    noseMat = glm::translate(noseMat, glm::vec3(0, 0, -0.3));
    noseMat = glm::rotate(noseMat, 0.15f, glm::vec3(0, 1, 0));
    noseMat = glm::rotate(noseMat, -rotAngleLeg * 35.0f, glm::vec3(0, 0, 1));
    scaleMat = glm::scale(identityMat, glm::vec3(0.225, 0.225, 0.4));
```

```
    pvmMat = projectMat * viewMat * worldRotMat * bodyMat * headMat * noseMat *
scaleMat;
    glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
    glDrawArrays(GL_TRIANGLES, 0, NumVertices);
}
```

머리의 모든 요소들은 모두 headMat 매트릭스의 영향을 받도록 설계하였다. headMat에 회전 transform을 적용하여 코끼리가 걸을 때, 머리 전체를 좌우로 흔드는 자연스러운 모습을 표현하였다. 각 모델의 모양과 위치를 만들어주는 modelMat과 scaleMat에 각 적절한 transform이 적용되도록 구성한 뒤, 일련의 pvmMat = projectMat * viewMat * worldRotMat * bodyMat * headMat * modelMat * scaleMat; 매트릭스 연산을 통해 최종적으로 tranform을 마치는 구조이다. 이때, 코의 경우 각 코 부위 별 계층이 존재하므로, 기존 transform 매트릭스에 연속적으로 transform을 진행할 수 있도록 코드를 구성했다. 또한 코의 경우에도 회전 transform을 적용하여 코끼리의 머리가 흔들릴 때, 코들이 조금씩 돌아가는 자연스러운 모습을 표현하였다. 이때, 코는 계층구조가 존재하여, 하단 계층의 코가 더욱 많은 회전이 되도록 애니메이션을 구현했다.

아래는 다리를 구성하는 코드이다.

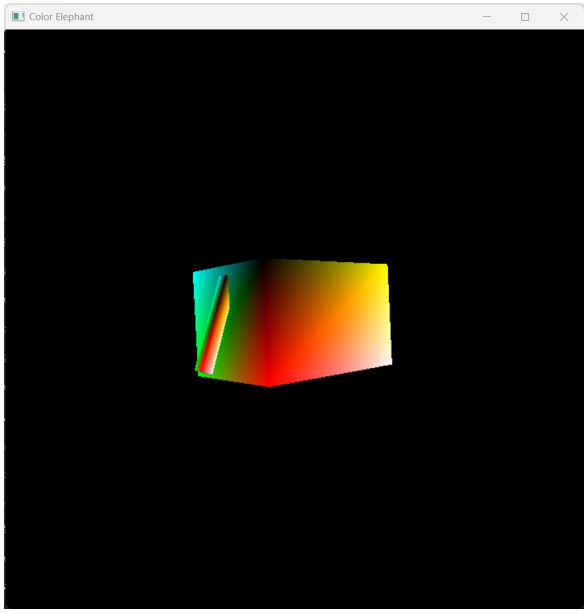
```
void drawLeg(glm::mat4 worldRotMat, glm::mat4 bodyMat)
{
    glm::mat4 modelMat, pvmMat, scaleMat, identityMat = glm::mat4(1.0f);
    glm::mat4 legMat;
    glm::vec3 eachLegPos[4];

    int eachLeglDir[4];
    const float xPos = 0.6f, yPos = 0.4f, zPos = 0.4f;

    eachLegPos[0] = glm::vec3(xPos, yPos, -zPos);   // rear right
    eachLegPos[1] = glm::vec3(xPos, -yPos, -zPos);  // rear left
    eachLegPos[2] = glm::vec3(-xPos, yPos, -zPos);  // front right
    eachLegPos[3] = glm::vec3(-xPos, -yPos, -zPos); // front left
    eachLeglDir[0] = 1;
    eachLeglDir[1] = -1;
    eachLeglDir[2] = -1;
    eachLeglDir[3] = 1;

    for (int i = 0; i < 4; i++)
    {
        // 허벅지
        legMat = glm::translate(identityMat, eachLegPos[i]);
        legMat = glm::translate(legMat, glm::vec3(0.0f, 0.0f, 0.5f)); // 상단 끝 부분을
회전 축으로 이동
        legMat = glm::rotate(legMat, -rotAngleLeg * 60.0f * eachLeglDir[i], glm::vec3(0,
1, 0));
        legMat = glm::translate(legMat, glm::vec3(0.0f, 0.0f, -0.5f)); // 다시 원래 위치로
되돌리기
        scaleMat = glm::scale(identityMat, glm::vec3(0.5, 0.5, 0.5));
        pvmMat = projectMat * viewMat * worldRotMat * bodyMat * legMat * scaleMat;
        glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
        glDrawArrays(GL_TRIANGLES, 0, NumVertices);

        // 무릎
```

```
        modelMat = glm::translate(legMat, glm::vec3(0, 0, -0.25));
        scaleMat = glm::scale(identityMat, glm::vec3(0.45, 0.375, 0.2));
        pvmMat = projectMat * viewMat * worldRotMat * bodyMat * modelMat * scaleMat;
        glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
        glDrawArrays(GL_TRIANGLES, 0, NumVertices);

        // 종아리
        modelMat = glm::translate(modelMat, glm::vec3(0.0, 0.0, -0.31));
        modelMat = glm::translate(modelMat, glm::vec3(0.0f, 0.0f, 0.5f)); // 상단 끝
부분을 회전 축으로 이동
        modelMat = glm::rotate(modelMat, -rotAngleLeg * 50.0f * eachLeglDir[i],
glm::vec3(0, 1, 0));
        modelMat = glm::translate(modelMat, glm::vec3(0.0f, 0.0f, -0.5f)); // 다시 원래
위치로 되돌리기
        scaleMat = glm::scale(identityMat, glm::vec3(0.35, 0.35, 0.5));
        pvmMat = projectMat * viewMat * worldRotMat * bodyMat * modelMat * scaleMat;
        glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
        glDrawArrays(GL_TRIANGLES, 0, NumVertices);

        // 발
        modelMat = glm::translate(modelMat, glm::vec3(0.025, 0.0, -0.25));
        modelMat = glm::rotate(modelMat, -rotAngleLeg * 75.0f * eachLeglDir[i],
glm::vec3(0, 1, 0));
        scaleMat = glm::scale(identityMat, glm::vec3(0.5, 0.36, 0.175));
        pvmMat = projectMat * viewMat * worldRotMat * bodyMat * modelMat * scaleMat;
        glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
        glDrawArrays(GL_TRIANGLES, 0, NumVertices);
    }
}
```

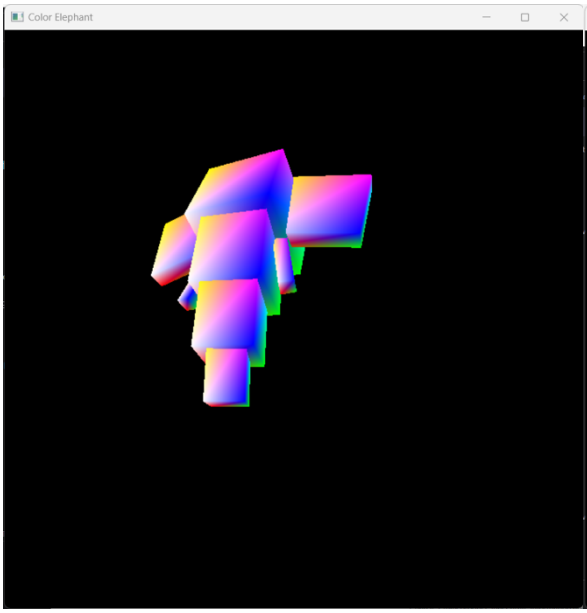
각 다리의 pos와 회전 방향을 결정해주는 배열들을 정의한 뒤, 허벅지-무릎-종아리-발 순으로 구성해 주었다. 모든 요소는 부모의 transform에 영향을 받을 수 있도록 행렬 곱을 진행하였다. 특히 다리의 경우 회전하는 축이 끝 쪽이어야 하므로 rotate를 전 후로 translate를 축에 맞추어 진행한다. 허벅지, 종아리, 발 3가지 요소가 각각 회전 하도록 구현하였다.

3. 결과 이미지

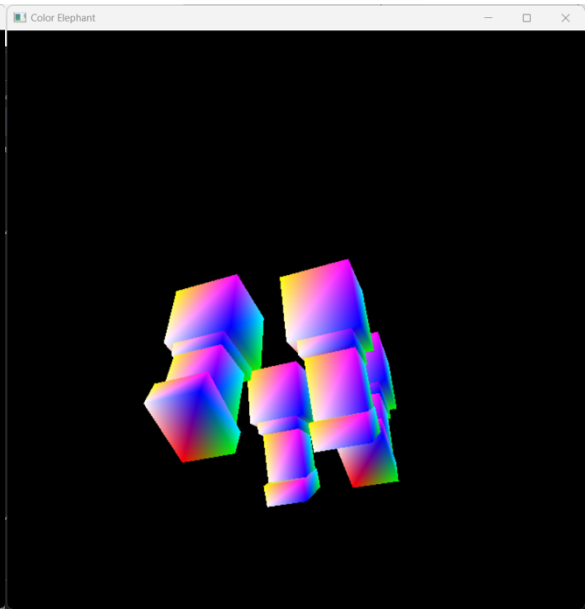
각각 몸통, 머리, 다리를 그리는 함수를 독립적으로 호출했을 때의 이미지이다.



(몸)

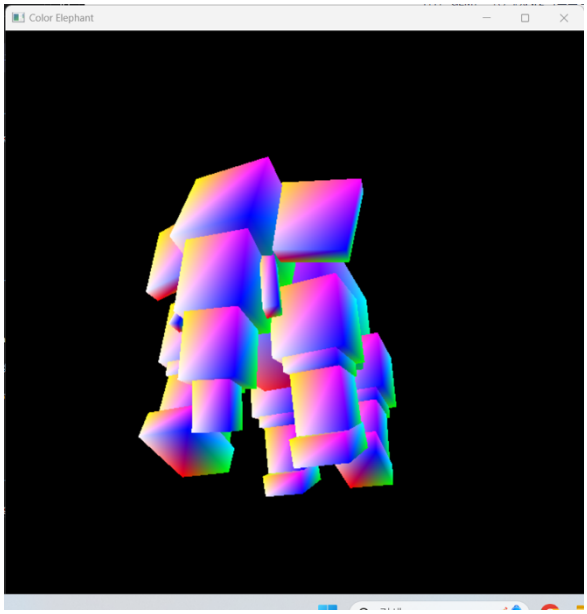


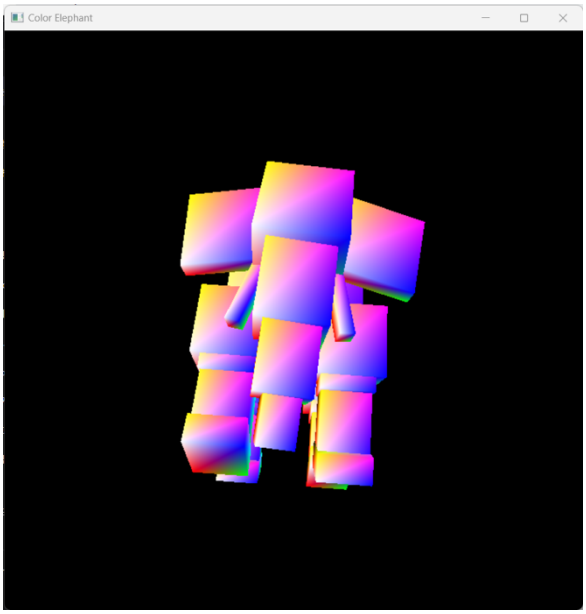
(머리)

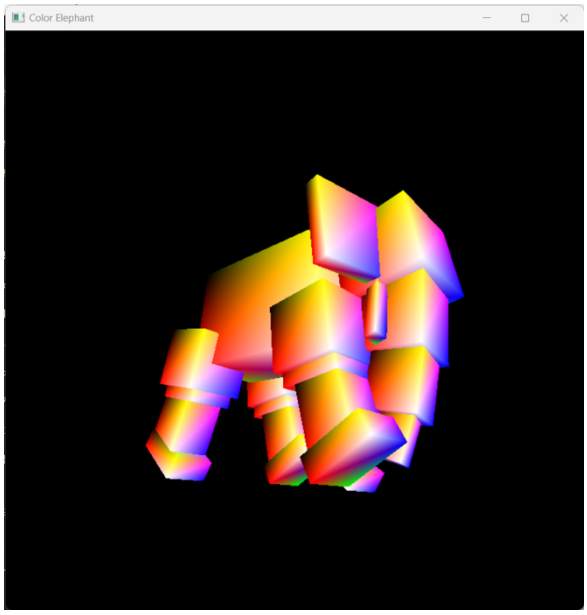


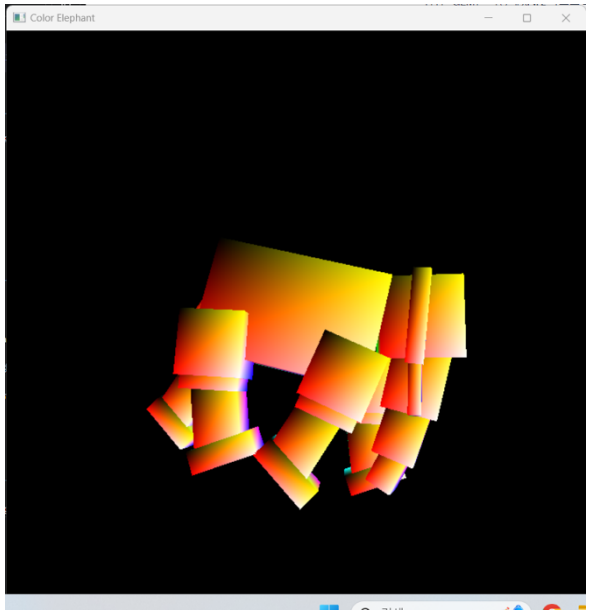
(다리)

모든 함수를 호출하여 코끼리를 그린 모습은 아래와 같다.









2

https://github.com/minseok128/OpenGL_Elephant