Student ID: 2016024893

Name: 오성준

1. **Introduction to the program, a brief description of how to run it, and how to use it.**

   이 프로그램은 손모양의 main으로 평면 위에 있는 공들을 모두 밀어내면, 감사인사와 함께 성공했다는 의미로 보라색의 주먹쥔 손이 튀어나오는 일종의 게임입니다. 미션을 완수해 보세요!.

   우선 A로 왼쪽으로 이동, D로 오른쪽으로 이동, W로 앞으로 이동, S로 뒤로 이동 가능하며, R로 왼쪽으로 돌 수 있고, T로 오른쪽으로 돌 수 있습니다. 바로 반대편을 보고 싶으면, E를 눌러 주세요. 손으로 공을 치게 되면, 공이 날아가게 됩니다. 공들을 모두 흰색 선들의 영역 밖으로 내보내 보세요.

2. **Description of the implementation of each requirement:** 1) How you implemented that requirement, 2) How TA can verify that the requirement has been implemented, by performing a program's specific feature.. **Requirement items not implemented should be left blank.**

   A. (15 pts) Draw 3 or more 3D objects.

   저는 총 5개의 3d object를 그렸습니다.

   1) main: main은 drawcube를 이용해서 draw main함수에서 그렸습니다. 전반적인 형태는 제가 만든 첫번째 과제의 모양과 비슷하지만, drawCube함수에는 glNormal3f를 사용했고 밑에서 설명하겠지만, glMultiMatrixf함수를 통해서 B번 명세를 구현했습니다.

   2) 공 3개: 2번째 과제 코드를 재활용했고, 마찬가지로 예시로 나왔던, sphere-tri object file을 써서 구현했습니다. Square 함수를 main에서 render를 호출하기 전에, 무한 루프를 돌기 전에 사용해서 자동 로딩 했습니다. 그러고 나서 draw_square함수를 써서 나타나게 했습니다. 각각의 공은 빨간색, 초록색, 파란색을 가집니다. 각각 위치를 나타내는 place 값과 방향을 나타내는 direct값이 존재합니다.

   3) final Hand: 게임을 다 끝냈거나, M키를 누르면 나오는 손입니다. 이 손은 끝을 표시하기 위한 것으로 final hand가 나옴과 동시에 감사 메시지도 나옵니다. Hand obj파일을 사용했고, final_hand함수를 main에 square처럼 선언해서 미리 로딩한 뒤에, draw hand를 통

해서 나타나게 했습니다. 보라색입니다.

B. (15 pts) One of the objects should be "main object". The user should be able to transform the main object using the mouse or keyboard.

Key callback만을 사용합니다.

1) W: x축으로 0.5만큼 translate 합니다.

2) S: x축으로 -0.5만큼 translate 합니다.

3) D: z축으로 0.5만큼 translate 합니다.

4) A: z축으로 -0.5만큼 translate 합니다.

5) R: 한번 입력에 5도씩 y축을 중심으로 rotate합니다.

6) T: 한번 입력에 -5도씩 y축을 중심으로 rotate합니다.

7) E: y축을 중심으로 reflect합니다.

8) T: 각 축에 대해, 1.1배만큼 scale합니다.

9) O: 각 축에 대해, 0.9배만큼 scale합니다.

10) I: x,z값을 y값을 이용해서 0.1만큼만 shear해줍니다.

위의 구현을 모두, 하나의 행렬에 곱하면서 누적시키고, 이 값을 place라는 main의 중심좌표에 곱해줍니다. 모두 오른쪽에 곱해서, local frame에 관하여 적용되도록 했습니다.

프로그램의 완결성을 위하여 repeat일 때는 작동하지 않습니다. 수고로우시더라도 하나씩 눌러서 확인해주세요. 감사하고 죄송합니다.

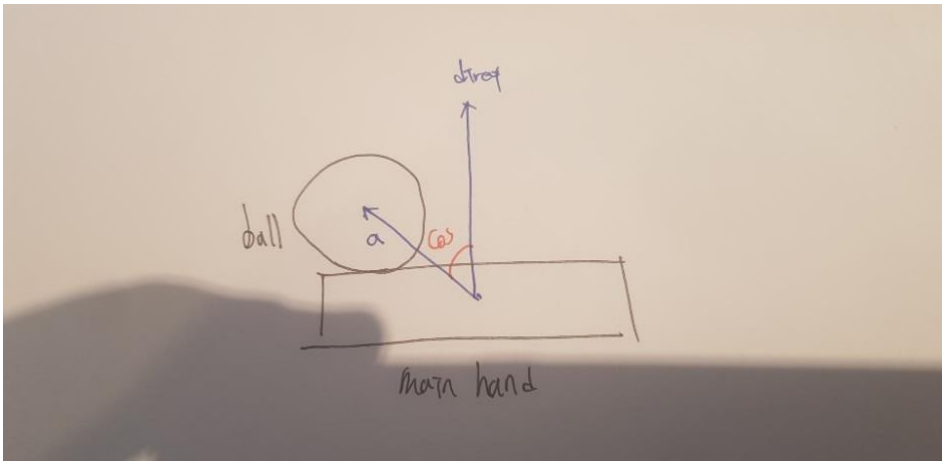
C. (15 pts) The rest of the objects should be automatically moved in response to the movement of the main object or other objects around it.

2가지의 구현요소가 있습니다. 첫번째는 공과 main인 손이 일정 거리 이상 가까워졌을 때, (부딪쳤을 때), 그리고 공과공이 일정 거리 이상 가까워졌을 때 입니다. 각각의 상황에서의 구현을 설명 드리겠습니다. 참고로 마지막에 게임이 끝난 후 나오는 손은 말 그대로 게임이 끝났기 때문에 불필요해서 구현하지 않았습니다.

참고로, 공은 손바닥을 맞지 않았을 때, 움직이지 않습니다. 괜히 손등으로 쳐보시지 않기를 바랍니다.

그리고 가급적이면 공을 치는 직전만큼은 한템포 쉬고 치시는 걸 추천드립니다. 빠르게 눌러서 안되는건 아니지만, 약간의 delay가 생길 수 있습니다.

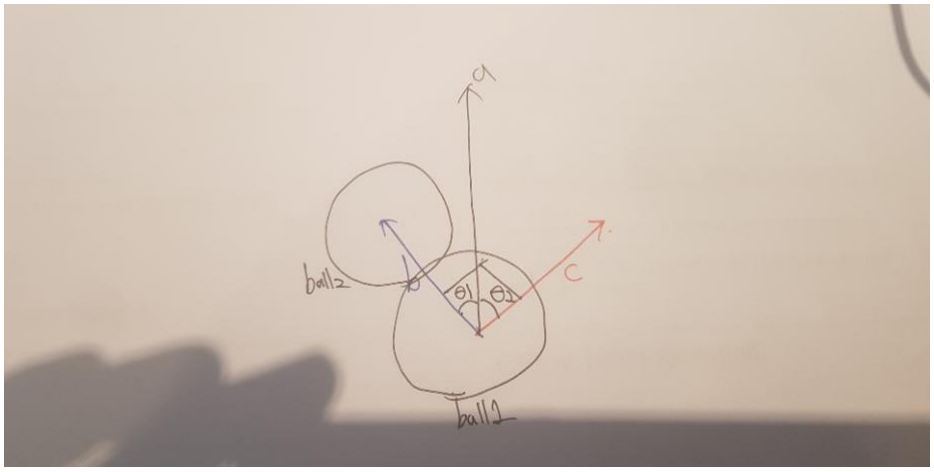
1) 공과 main hand가 부딪쳤을 때,



공과 손이 부딪치면, 손은 움직이지 않고 공만 날라가다록 설정되있습니다. 다만 이 때, 공이 날라가는 방향은 공의 중심과 손의 중심의 차이로 벡터를 만들어서 그 방향으로 일정 수준 날라가도록 했습니다. 이 때, 정면에 있을 때가 가장 멀리 날라가도록 만들기 위해서, cos값, 즉 사이각을 구하고, 그 값으로 cosin 값을 구해서, 그 값을 곱해줬습니다. A와 direct벡터를 내적해서 구했고, 손의 어느면에 따라서 얼마나 움직이는지 결정됩니다. 그리고 cosin(cos)값을 0과 1사이일때만, 공이 날라가도록해서, 적어도 손바닥 안에 맞아야 움직이도록 만들었습니다.

공이 움직일 때는, 충돌 이후 각 공의 번호에 맞는 moving signal이 true가 되야 움직일 수 있으며, 목표지점을 after로 해서 place와 linspace해서 100개 정도로 나눈 뒤, global 변수인 lin_cnt가 0부터 99까지 카운팅하면서, render마다 하나씩 그려주도록 해서 동작이 최대한 이어지도록 만들었습니다. 그 와 동시에 rotate함수를 써서 공이 굴러가는 걸 나타내도록 만들었습니다. 다 움직면 place값은 after값이 되고, moving은 false가 됩니다.

2) 공과 공이 부딪쳤을 때,



공과 공이 부딪쳤을 때는 좀 더 복잡합니다. 공과 공이 부딪칠때는 공 하나는 서있고, 다른 공 하나는 움직일 때가 있습니다. 이 때 만약 ball1이 a의 방향으로 움직이고 있었다면, ball2의 움직임은 자명하게 b의 방향일 것입니다. 위의 경우와 비슷하게 말입니다. 하지만, ball1의 움직임을 결정해야 했습니다. Ball1의 움직임은 c방향으로 원래 진행방향 a를 90도에서 theta1만큼을 뺀, theta2만큼 돌려서 c방향으로 가게 했습니다. 물론 왼쪽으로 부딪칠 때도 구현했는데, 이 때는 a와 b를 외적해서 양수일때와 음수일 때로 방향을 구분했습니다. 아마도 쉽게 c방향으로 움직이는 것을 보실 수 있으실 겁니다. 이 때는 원래 움직이던 공의 lin_cnt값을 이용해서 움직임의 가중치를 줬습니다. 왜냐하면, 늦게 맞으면 조금 움직이는게 상식이니까요. (100-lin_cnt)/100만큼의 가중치를 줬습니다. 그래서 그만큼 움직이게 만들었습니다. 그리고 원래 움직이던 공은 원래 가려하던 거리중 남은 거리 만큼을 그대로 움직이지 않고, 65프로정도만 움직이도록 했습니다. 원래는 운동량을 계산해야 하지만, 그냥 결과를 쉽게 볼 수 있는 방향으로 만들었습니다. 그리고 그 사이에 마찰력에 의한 에너지 손실은 제 능력 밖의 일입니다. 그래서 임의로 잡았습니다. 그 이후로는 moving=True가 되고 1번의 방식처럼 공들이 움직입니다.

확인하기 쉽도록 공들을 일직선 위에 올려놓은 점 참고 하시길 바랍니다.

각 상황에서 만약에 scale을 한다면, 부딪쳤다의 기준인 dist값을 p를 누르면 1.1배, o를 누르면 0.9배가 되도록 해서 손 크기에 영향을 받도록 만들었습니다.

D. (15 pts) Use perspective projection. The camera should be able to be switched between two modes:

V key를 누르면 1인칭 모드와 쿼터뷰를 왔다갔다 하실 수 있습니다. glPerspective함수를 사용해서, 명세대로 perspective projection을 사용했습니다.

1) 1인칭 시점의 구현은 glulookat함수에서 눈의 위치를 place 즉, main의 위치로 두고, 바로보는 위치를 place+direct 즉, 바로 main object가 W키를 통해서 진행하는 방향을 바라보도록 했습니다. w키는 통상적으로 앞을 나타냅니다. 그래서 선택했습니다. 어차피 place 값과 direct값은 B번에 있는 key를 누르면서 함께 바뀌기 때문에, 카메라가 함께 translate하고 rotate 됩니다.

2) Quarter view의 구현도, glulookat을 사용해서 했습니다. 바라보는 지점을 항상, place즉 main의 위치로 두고, 카메라의 위치를 place+25정도로 두어서 항상 main의 위치에서 (25,25,25)만큼 올라가서 main을 지켜보도록 했습니다. 계속 일정한 간격을 유지하기 때문에, panning이 구현됩니다.

E. (15 pts) Use OpenGL lighting / shading.

1) Lighting

빛 하나는 (-100.,-100.,-100.,1.)에서 흰색 빛을 비추는 빛이고 다른 빛 하나는

(np.cos(t)*25,25.,np.sin(t)*25,0.) 에서 t는 시간인데, 시간동안 animated하면서 보라색 빛을 만들어냅니다. 보라색 빛이기 때문에, 초록색 공을 보는 시선은 어둡지만, 대신에, 초록색 공은 흰색 빛을 비추는 부분은 굉장히 밝습니다.

2) Shading

Object file을 사용할 경우에는 object file에 있는 값을 따라서 glDrawArray로 shading했고, cube 같은 경우에는 ppt자료에 나와있는대로 glnormal을 사용해서 shading을 했습니다. 항상 빛이 두군데인 점을 염두해 주시면서 보셨으면 좋겠습니다.

각각의 object color는 위에서 설명했으니 넘어가겠습니다.

F. Extra credit

1) 제 object중에서 main hand는 항상 위아래로 손가락을 움직입니다. 애초에 hierarchical model과제에서 나온 것이기 때문에 문제가 없을 것이고, 사각형 하나를 삼각형으로 잘라서, polygon mesh를 구현했습니다.

2) M을 누르거나 게임을 마치면 나오는 hand는 p0 = np.array([0.,0.,0.])*7, p1 = np.array([2.,3.,1.])*7, p2 = np.array([2.,-3.,1.])*7, p3 = np.array([1.,0.,2.])*7 이 4개의 점을 bezier_cal을 사용해서 계산한 후 사용합니다. 이 때 동작은, 0과 1사이를 100개로 linspace해서, fish값을 계산하면서 render당 하나의 점에 해당하는 hand를 그리도록 하여, bvh 파일처럼 연속성이 드러나도록 했습니다. Fish는 99가 되면 0까지 작아지고, 0이 되면 99까지 커져서, 일정 구간을 왕복하도록 했습니다.

공을 1인칭 시점으로 놓으시고 약간 거리를 두시고 보시면 곡선 움직임을 쉽게 확인하실 수 있습니다.

각 공의 out변수를 만들어서 밖에 나갔는지 확인하고, 모두 나가면 보라색 손이 튀어나옵니다.