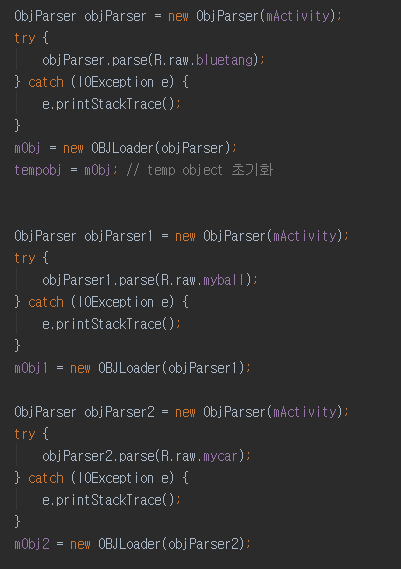
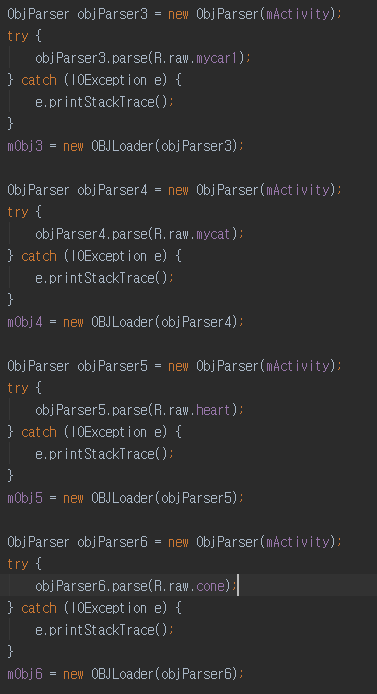
<모바일응용 소프트웨어 설계>

# 14주차 실습 과제

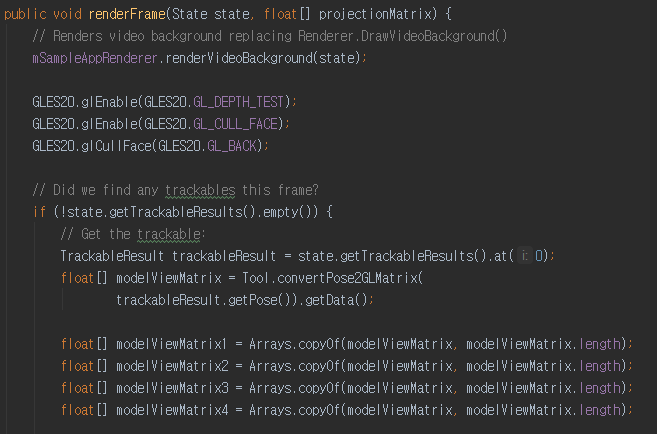
정보통신공학과

12161774 이 채 은

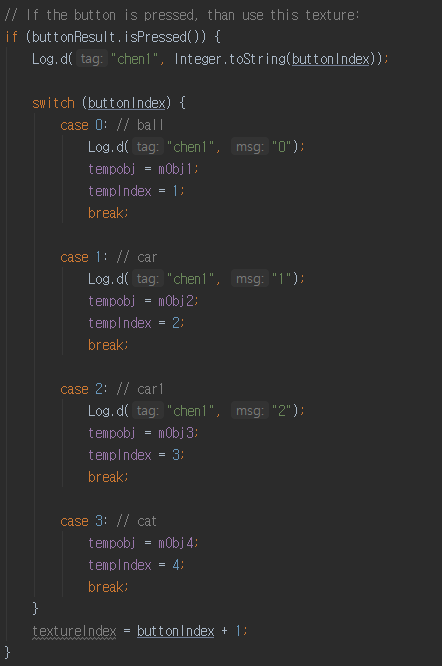
**[소스 코드]**

1. VirtualButtonRenderer.java

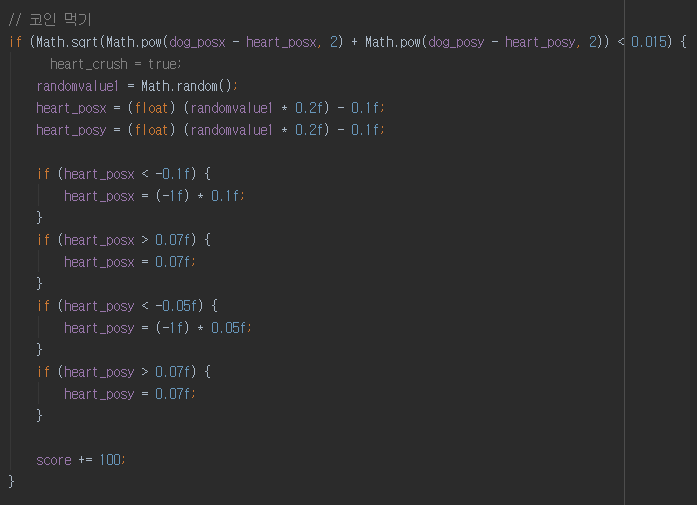
=> initRendering( ) 함수 내에서 objParser를 이용해서 object를 렌더링 해준다. ImageTargets에 jpg로 저장된 텍스처를 불러와서 R.raw.파일이름 으로 parser에 맞게 mapping 해줘야 한다. 또한 각 object 마다 따로 렌더링을 해야 하기 때문에 각 object 에 따라 0~6까지 코드를 따로 작성했다.



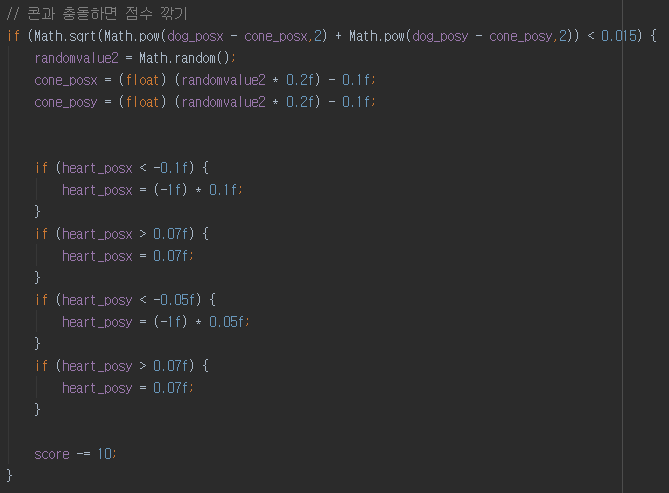
=> renderFrame 에서 modelViewMatrix를 초기화한다. 여러 개의 object를 화면에 띄우기 위해서 각각 다른 modelViewMatrix를 사용해야한다. 따라서 여러 개의 modelViewMatrix를 사용하기 위해서 Arrays.copyOf를 이용해서 원본의 lengh만큼 같은 크기를 갖는 modelViewMatrix1~4를 만들어준다.



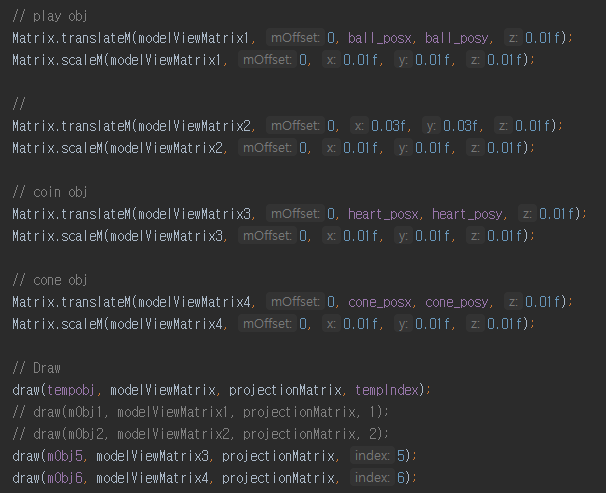
=> virtual button을 사용하기 위해서 각 virtual button이 눌렸을 때 경우를 나타내는 코드를 작성한다. If문을 이용해서 button 이 isPressed 상태일 때 swith문으로 누르는 버튼에 따른 상황을 정의해준다. 각 버튼은 button Index를 가지고 있기 때문에 버튼의 구분은 Index로 해준다. 첫 번째 버튼부터 순서대로 0 1 2 3 이다. 버튼을 누르면 화면에 띄워진 object가 바뀌는 것으로 구현했다. 따라서 맨 처음에 강아지가 로드 되어 있는데 첫 번째 버튼을 누르면 비행기로 바뀌고, 두 번째 버튼을 누르면 racing car, 세 번째 버튼을 누르면 빨간 차, 네 번째 버튼을 누르면 파란 차로 바뀐다. Temp object를 만들어서, 버튼이 눌릴 때 마다 새로 나타내고 싶은 object를 로드할 수 있게 했다.



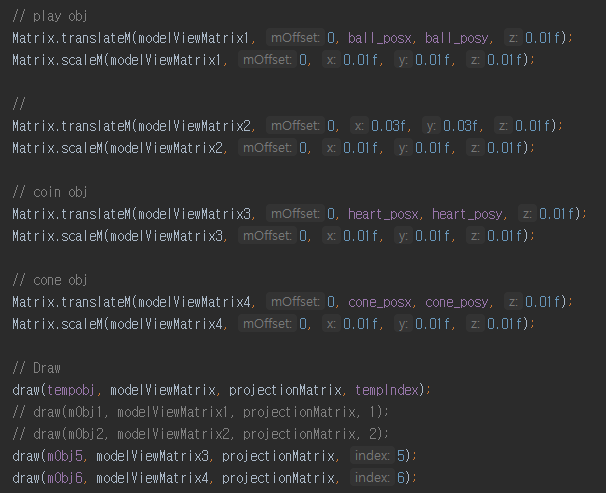
=> Objcet를 움직여서 코인을 먹으면 점수가 올라갈 수 있게 하는 코드이다. 코인과 object의 거리 차를 계산해서 특정 거리 이하이면 충돌한 것으로 파악하고, score 를 100점씩 올린다. Math.sqrt 와 Math.pow를 이용해서 각 object의 x좌표와 y좌표 사이의 거리를 비교한다. 이 때 coin 과 충돌하고 난 다음에는 새로운 위치에 다시 coin 을 위치 시키기 위해서 Math.random을 이용해서 랜덤 위치에 coin 이 생기도록 한다. 또한 화면에 보이는 곳 내부에 코인이 위치하도록 하기 위해서 if 문 4개를 이용해서 화면 끝의 범위를 설정했다.

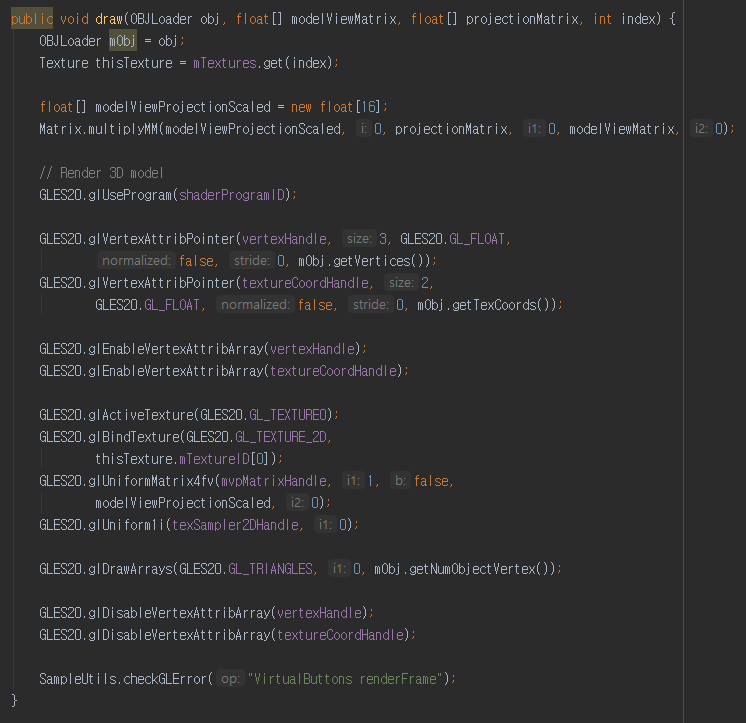


=> object가 콘(cone) 과 충돌하면 점수가 깎인다. 위의 coin을 랜덤 위치에 위치 시킨 것과 같은 원리로 랜덤 위치에 cone을 위치 시키고, 충돌 event 역시 두 object 사이의 x좌표와 y좌표의 거리 차이를 이용해서 계산한다.



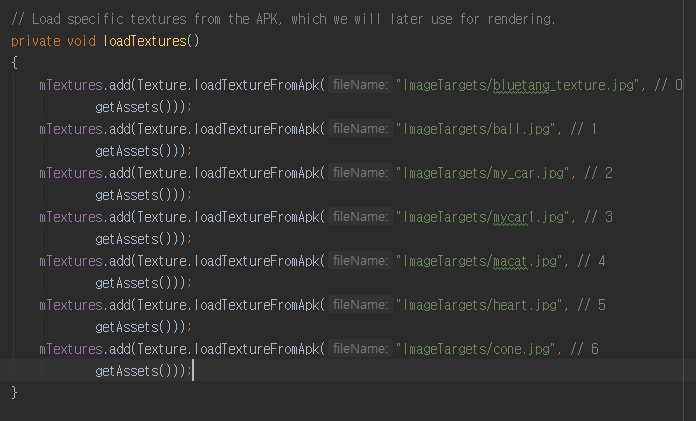
=> Object를 modeling transform 하기 위해서 각 object 별로 Matrix를 이용해서 translate, scaling 해준다. Translate는 위치를 이동하는 효과를 나타내고, scale은 회전 효과이다. 설정한 x, y, z 만큼 바뀌게 되고 바뀐 정보는 최종적으로 modelViewMatrix에 저장된다. 이 단계도 object별로 따로 해줘야 한다.



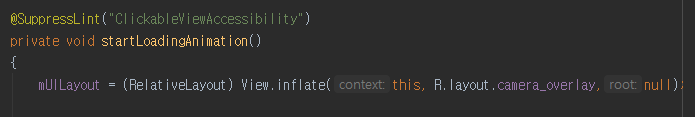


=> 여러 개의 object를 다루기 위해서 위와 같이 draw 함수를 만들어서, object를 화면에 띄울 때 draw 함수를 호출한다. 위에서 OBJLoader를 통해서 렌더링 했던 obj 명과, 최종 정보를 저장할 modelViewMatrix, 중간 과정에서 쓰이는 projectionMatrix, texture index를 불러올 Index 를 순서대로 넣어주면, draw 함수에서 렌더링을 완료하고 object를 모델링해서 화면에 띄워준다. Modelview projection 행렬은 modeling transform 의 모든 정보를 다 가지고 있다고 볼 수 있다. Tempobj 를 draw 할 때는, virtual 버튼에서 버튼에 따라서 다른 obj를 tempobj 에 넣어주기 때문에 다른 object가 draw 되는 것이다.

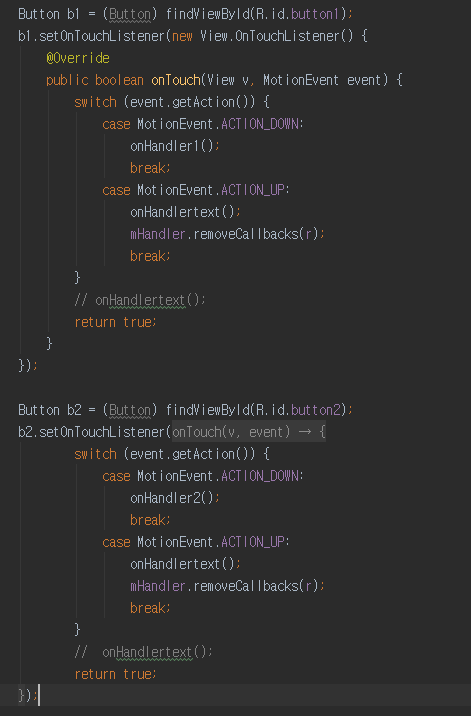
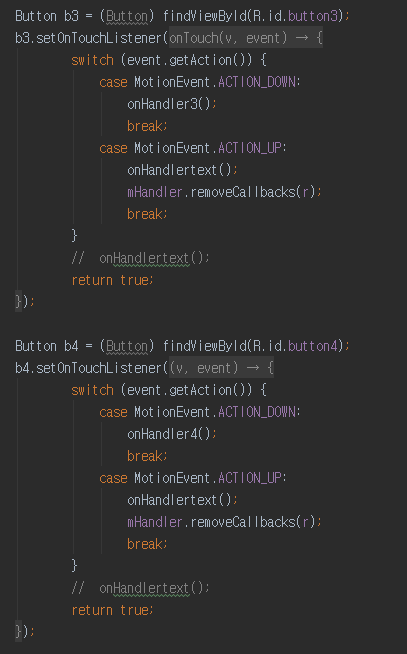
2. VirtualButtons.java



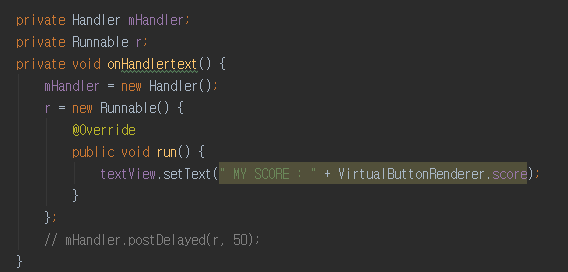
=> loadTexture 함수에서 각 object에 로드 할 텍스쳐 파일을 추가해준다. Object 별로 따로 texture mapping을 해주고, index도 순서대로 부여된다.



=> starLoadingAnimation 에서 camera\_overlay layout을 불러온다. 따라서 이 레이아웃에서 추가해준 textview와 버튼을 쓸 수 있다.



=> virtual 이 아닌 위치를 움직이는 버튼을 쓰기 위해서 정의해준 코드이다. Camera\_overlay 레이아웃에 정의된 버튼을 불러와서 left, right, up, down으로 움직일 수 있도록 한다.



=> Handler 를 등록해서, text 내용을 바꿔줄 수 있도록 한다. Handler mHadler를 초기화해주고, 앱이 구동되는 동안 바뀌는 내용을 textView에 띄워준다. Object를 조정해서 coin을 먹으면 점수가 100점 씩 올라가고, 장애물인 cone과 충돌하면 점수가 10점씩 깎인다. 따라서 이 점수를 virtualButtonRenderer에서 score로 static 으로 선언했기 때문에 불러와서 textView에서 보여준다.

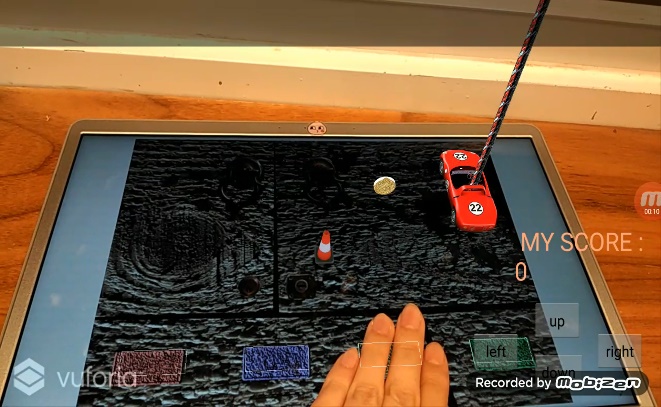
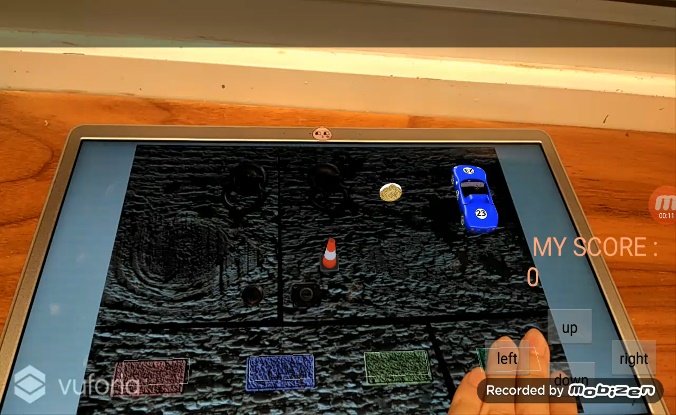


=> 버튼의 구현 내용을 Handler를 통해서 정의해준다. 각 버튼에 맞는 핸들러가 있고, run 될 때 버튼의 기능에 따라서 내용이 변하게 된다. 아래로, 위로, 양 옆으로 움직이는 것을 구현하기 위해서 화면에 떠있는 object의 위치를 pos\_x, pos\_y 의 x좌표, y좌표를 통해서 바꿔준다. 이는 렌더러에서 Matrix translate에서 위치를 변화시킬 때 이용된다. 마찬가지로 angle 값을 0, 90, 180, 270도로 바꿔주면서 한 바퀴를 돌 수 있도록 한다. 이 역시도 렌더러에서 Matrix rotation 에서 이용된다.

**[ 실습 결과 및 고찰 ]**



=> 맨 처음 렌더링 됐을 때 상황이다. 화면에 기본 object인 강아지와, coin, cone 이 렌더링 되어있고, virtual button 과 MY SCORE 이 떠있다.



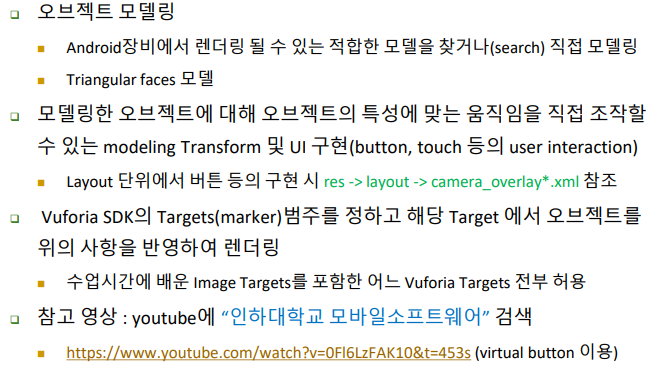
=> virtual button 을 누를 때, 각 버튼에 따라서 object가 변한다. 첫 번째 버튼을 누르면 강아지에서 비행기로, 두 번째는 레이싱카로, 세 번째는 빨간 범퍼카로, 네 번째는 파란 차로 바뀌게 된다. 따라서 원하는 대로 virtual button 을 눌러서 object를 변화시킬 수 있다.



=> 화면의 오른쪽 아래에 떠 있는 버튼을 눌러서 상하좌우로 object를 이동시킬 수 있다. 이 때 coin 과 충돌한 경우 MY SCORE은 100점 상승하고, coin 은 다시 새로운 위치에 위치하는 것을 확인할 수 있다.



=> object를 이동 시키다가 cone 과 충돌하는 경우에는 점수가 10점 깎인다. 따라서 MY SCORE이 10점 깎여서 나타나는 것을 확인할 수 있었다. 또한 충돌 이후에는 cone이 새로운 위치에 랜덤으로 위치하게 된다.



=> 이번 과제를 통해서, 내가 원하는 Target에서 내가 원하는 object를 직접 렌더링하고 화면에 띄워서 원하는 대로 움직이는 것까지 구현해봤다. Object를 모델링하기 위해서 렌더링 될 수 있는 적합 모델을 직접 찾고, blender를 이용해서 obj 파일을 import 하고 triangular faces 모델로 export했다. 또한 맞는 obj 파일과 텍스처 파일을 함께 mapping 해서 object위에 원하는 텍스처를 입히는 과정도 진행했다. 이렇게 렌더링한 object를 원하는 위치에 띄우고, 원하는 위치와 방향으로 이동하고 회전시키기 위해서 matrix를 이용했는데 matrix.translate, matrix.ratate를 이용해서 modelViewMatrix에 이 내용이 저장된다. 그리고 이 행렬을 이용해서 최종적으로 modelig transform 을 완료하고 화면에 띄우게 된다. 원하는 위치와 회전 각도를 변수로 직접 지정하고 행렬을 정의하는 과정에서 값을 넣어주기 때문에 원하는대로 화면에 object를 띄울 수 있었다. initRendering 에서 parser 를 이용해서 각 객체의 obj 에 texture를 mapping 해서 렌더링 시킬 수 있었다. 또한 이번 과제에서는 모델링한 object에 대해서 user interaction을 구현하기 위해서 virtual button을 만들었다. Layout 상에 있는 button이 아니라 내가 등록한 target에서 직접 버튼을 만들어서 가상으로 버튼이 생기고, 그 부분을 눌러서 press 상태가 되면 레이아웃에 있는 버튼을 누른 것 처럼 object에 대한 새로운 event 를 구현할 수 있다는 점이 신기했다. 하지만, 예제에서 주어진 java 코드에서는 image targeting 만 진행할 수 있었기 때문에 virtual button을 같이 구현하기 어려웠다. 하지만, 새로운 virtualbutton java 파일이 있는 것을 확인했고, virtualbutton Renderer 에서 renderFrame 에서 버튼에 맞는 내용을 구현해서 사용할 수 있음을 알게됐다. 실제로 앱을 실행시켰을 때 버튼을 누르면 object가 변하는 것을 확인했을 때 정말 신기했다. 하지만, 가상으로 만들어진 버튼이기 때문에, target 에서 특별한 색을 인식하고 버튼으로 인식하는 것이다. 따라서 정교하게 눌러졌을 때 제대로 버튼이 눌린것과 같은 효과가 구현됐다. Target 이 전체 다 나오지 않고 가려진 경우나, 흔들리는 경우 virtual button이 제대로 구현되지 않아서 주의해야 했다. 그리고 레이아웃에서 만든 버튼으로는 object를 움직이는 데 썼다. 따라서 상, 하, 좌, 우로 어느 정도씩 translate 해줄지, 어느 각도씩 rotate 해줄지 설정해주었다. 마지막으로 object를 화면에 띄울 때 draw 함수를 이용해서 여러 개의 object도 한번에 띄울 수 있도록 했는데, 이 경우에 modelviewmatrix를 각 object별로 따로 만들어주어야 했다. 위의 과정에서 사용자가 설정한 대로 object를 움직이기 위해서 modeling transform 을 했는데 위의 최종 결과는 modelviewmatrix에 저장되기 때문이다. 따라서 이번 과제를 통해서 object 를 렌더링 하는 것부터, 내가 원하는 target을 데이터로 등록하고 쓰는 것, user interface에 맞게 조정하는 것까지 직접 실습하고 배우고, 앱으로 구현해 볼 수 있었다.