GRAPHICS

HW1

제출일 : 2019.04.17

학번 : 20171690

이름 : 정유석



Release, x86 모드에서 정상적으로 컴파일되고 작동하는 것을 확인하였습니다.

1. **SWORD TRANSFORMATION** – rotation, translation, scaling

rotation과 translation, scaling transformation을 이용해 구현하였다. WINDOW의 width와 height중 작은 것을 지름으로 하는 원의 둘레에서 원점을 향하여 sword object가 위치한다. 이 때 sword는 30 degree 간격으로 총 12개를 그린다.

일정한 시간 간격마다 sword들은 원점으로 translation 변환되며, 같은 시간 간격마다 sword의 크기가 줄어드는 scaling 변환이 일이난다. 이 때 각 sword는 초기의 원 둘레에서 크기가 3.5배로 scale 되며 원점에서는 0이 되도록 일정 간격으로 줄어든다.

한편 window size가 변결될 경우, radius 값도 그에 따라 변경되며 처음과 마찬가지로 WINDOW의 width와 height 중 작은 값을 지름으로 하여 반지름을 저장한다.

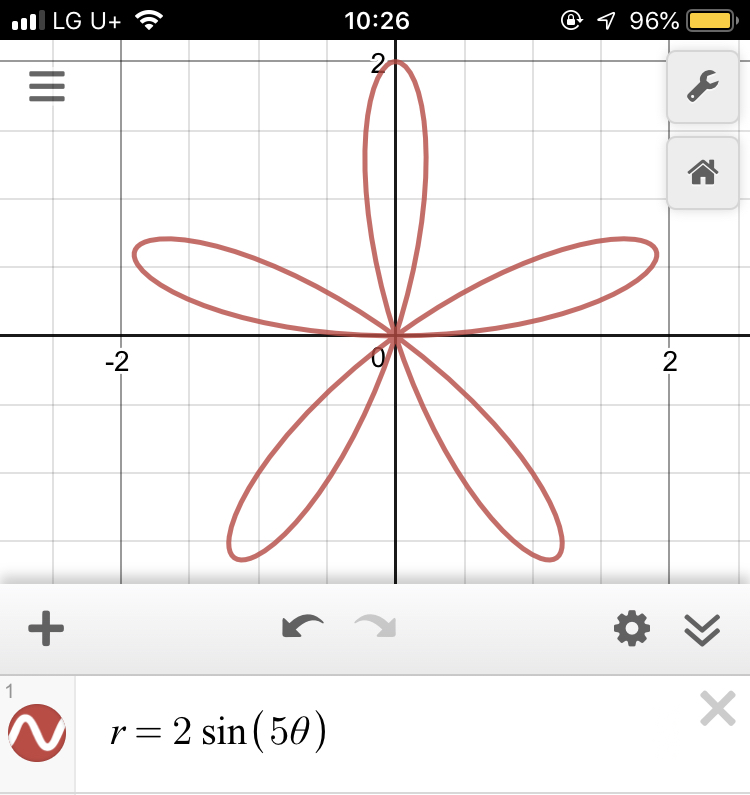
1. **HOUSE TRANSFORMATION** – scaling, shearing

간단한 shearing transformation을 구현하였다.

원점에 위치한 house object가 차례대로 y축 2배 scaling, x축 shearing, x축 2배 scaling, y축 ½ 배 scaling, x축 ½배 scaling transformation이 일어난다.

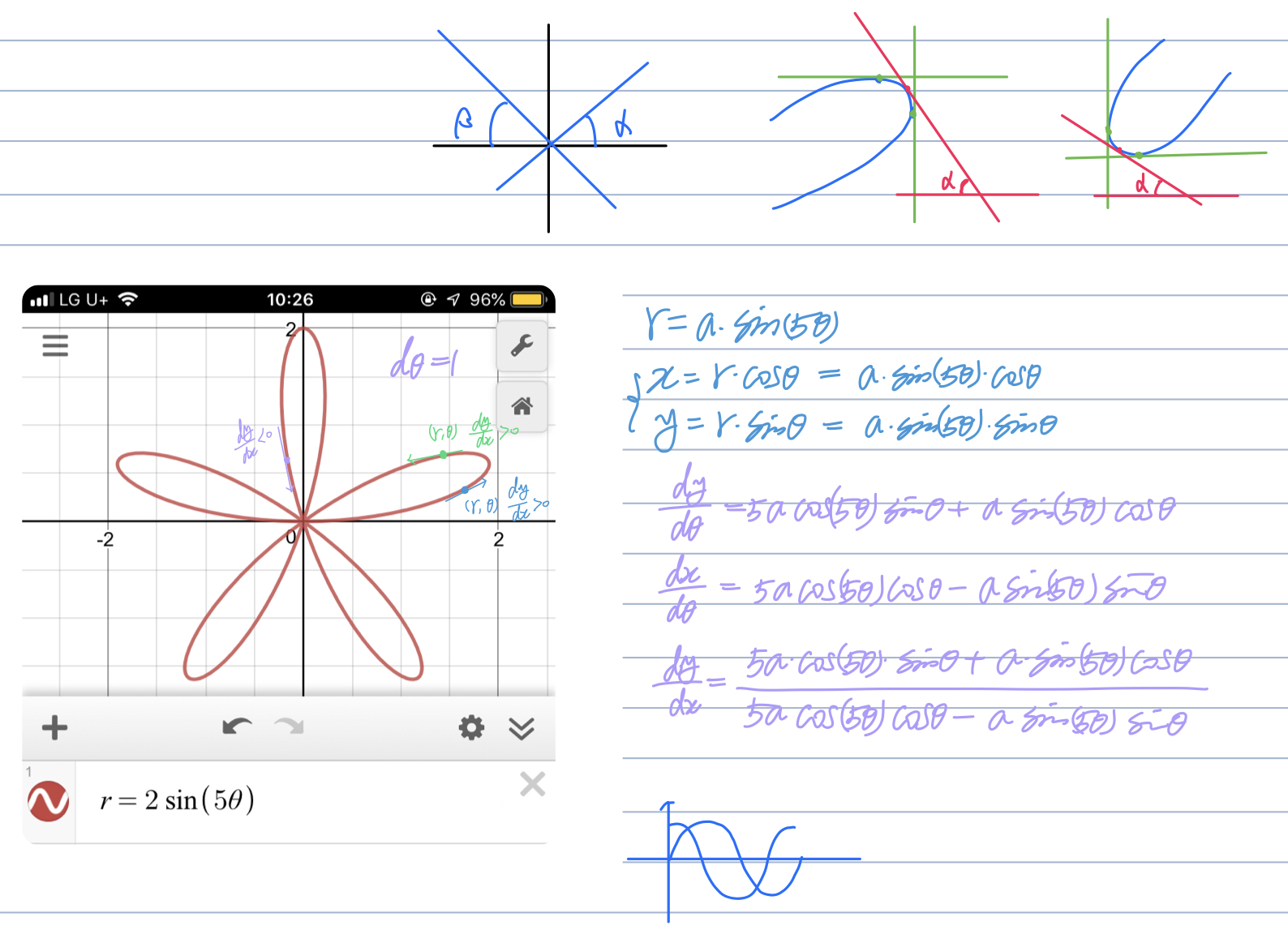
1. **AIRPLANE TRANSFORMATION** – translation, rotation

아래와 같은 극좌표계의 점을 이동한다.



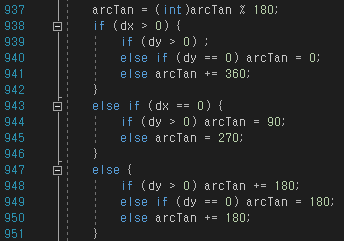
이 때 식의 방정식은 r = radius \* sin(5 \* theta) 가 되며, x = radius \* r \* cos(theta), y = radius \* r \* sin(theta)로 xy 좌표계로 이동시켜 translation transform을 구현하였다.

한편 위의 그래프에서 각 점에서 theta에 관하여 미분한 값을 이용해 기울기를 이용하여 airplane object의 방향을 적절히 설정하였다.

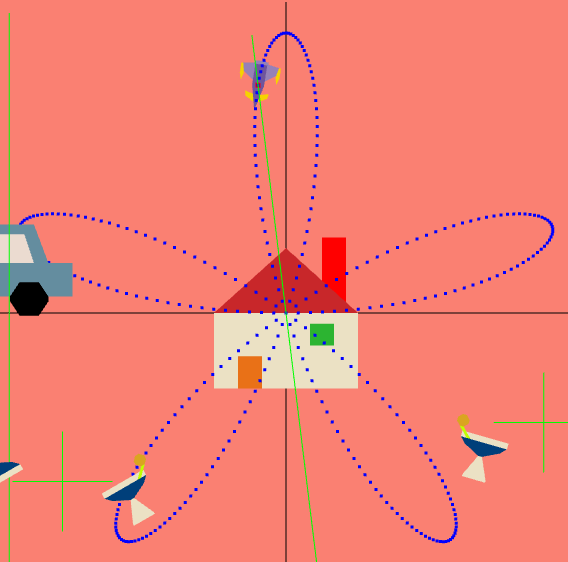


위의 그림에서와 같이 theta에 따라 변하는 x의 값과 y의 값인 dx, dy를 계산하여, 그 값을 이용해 각 점에서의 기울기를 계산하였다. 이 기울기를 이용하여 접선이 x축 양의 방향과 이루는 각을 계산할 수 있었으며, 최종적으로 이를 이용해 airplane object의 방향을 결정하고, rotation transformation을 적절히 적용할 수 있었다.

한편 접선의 기울기를 이용하여 x축 양의 방향과 이루는 각을 계산하는 과정에서, arc tangent의 결과값은 -90 degree 초과 90 degree 미만이 된다. 따라서 x와 y의 변위인 dx, dy 값을 적절히 활용하여 접선의 각을 0 degree 이상 360 degree 미만의 값으로 변환시켰다. 이를 수행하는 코드는 아래와 같다.



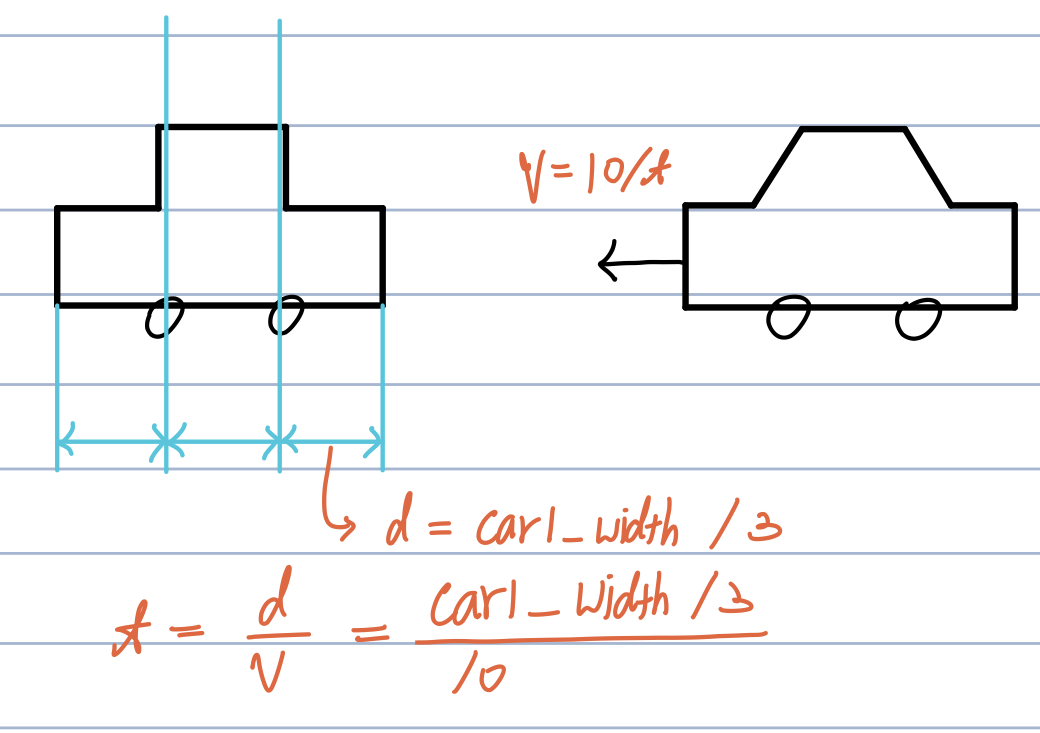
한편 window size가 변할 때마다 그려지는 비행기가 움직이는 경로의 도형 크기는 달라진다. 또한 키보드에서 ‘l’ 또는 ‘L’ 키를 입력했을 때 비행기가 이동하는 경로(위 좌표계에서의 그래프)를 점을 이용하여 화면에 그린다. (아래 사진의 파란 점)



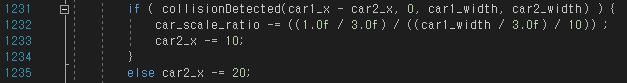
1. **CAR TRANSFORMATION** – translation, scaling, rotation

translation, scaling, rotation transformation을 활용해 두 자동차가 충돌하여 한 자동차가 찌그러진 후 날아가는 모습을 구현하였다.

이 때 car2가 이동하다가 car1과 만나는 순간, 즉 충돌이 일어나는 순간 car2의 이동 속도는 이전보다 상대적으로 느려진다. 또한 충돌이 있고 car1의 width가 1/3이 될 때까지 car1은 x축 기준 scaling이 된다. 이 때 간단한 물리를 이용해 두 자동차가 충돌하여 이동하는 시간 및 거리와 그에 따른 car1의 scaling 비율을 구할 수 있다.

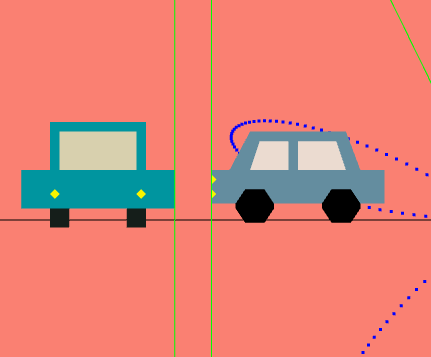
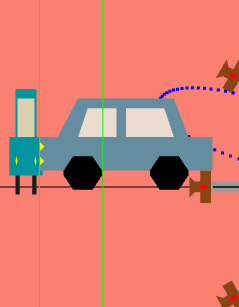


위의 그림에서 두 자동차는 충돌 후 거리 d만큼 t 시간동안 이동한다. (이 때 car2의 속도 v는 충돌 후 속도인 10을 의미한다.) 따라서 car1이 x축 기준으로 scale되는 비율은 (1/3) / t가 되며, 이를 코드로 나타내면 아래와 같다.



car1의 width가 처음의 1 / 3이 된 후, car2는 정지하고 car1 화면의 어떤 직선을 따라 날아간다. 이 때 직선의 기울기는 -3 이상 3 이하가 되며, 처음에 car2가 날아가는 방향은 왼쪽 방향이다.(x 좌표가 감소하는 방향) 그리고 car2는 벽(Window 4면)을 최대 2번 부딪히고, 마지막 경로에서 화면 밖으로 이탈한다. car2가 날아갈 때 car2는 빠른 속도로 360 degree 돌아간다.

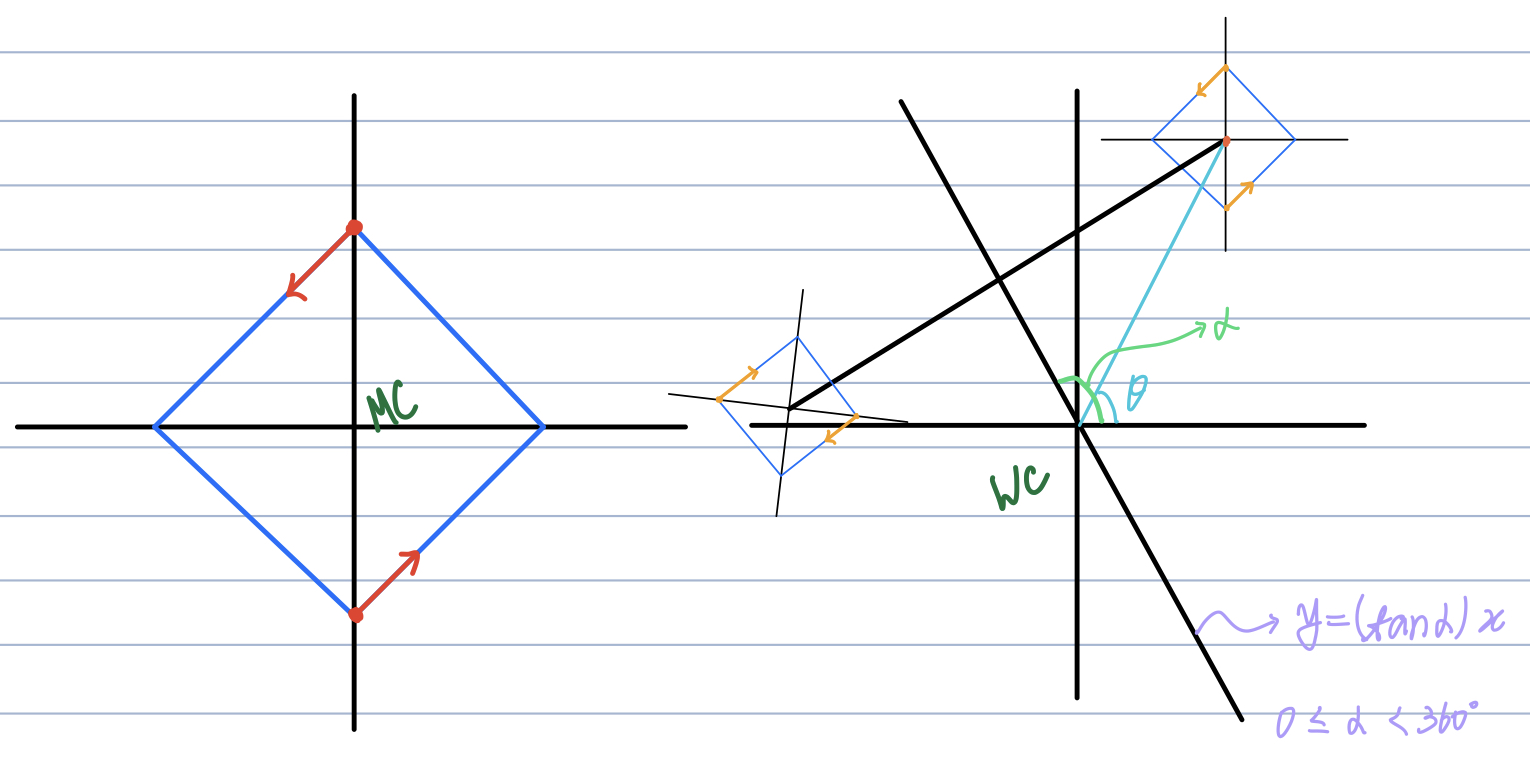
‘l’ 또는 ‘L’ 키보드를 눌렀을 때, car1의 (scale되기 전) 가장 오른쪽 성분을 지나는 y축에 평행한 선분과 car2의 가장 왼쪽 성분을 지나는 y축에 평행한 선분을 확인할 수 있다.

< 각각 두 자동차가 만나기 전과 만난 후 해당 선분의 위치 >

1. **COCKTAIL TRANSFORMATION** – translation, rotation, reflection

cocktail object에 대해 translation, rotaion, reflection transformation을 적용하여 복잡한 기하변환을 구현하였다.



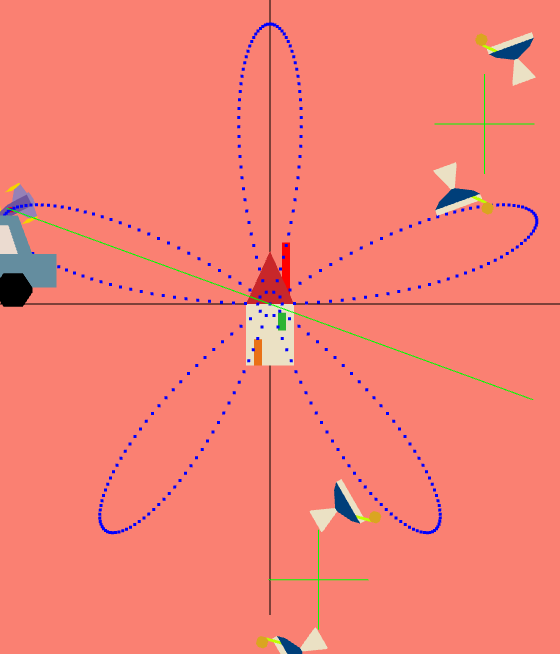
cocktail object 하나는 위 그림 왼쪽 좌표계의 y 축 위의 한 점에서 출발한다. 이 때 출발점의 y 좌표는 양수이다. 출발점을 출발하여 매순간 빠른 속도로 360 degree 회전하며 다이아몬드 모양을 그리며 이동한다. 이러한 이동을 원점 대칭 시킨다. 그러면 2개의 cocktail object가 원점을 기준으로 왼쪽 그림의 파란색 경로대로 이동한다.

이 때 두 칵테일이 이동하는 이 좌표계를 새로운 Modeling Coordinate라 하자. 위 그림의 왼쪽 좌표계를 의미한다. 실제 화면, 즉 World Coordinate에서는 이 새로운 MC를 WC의 원점을 기준으로 반지름이 radius인 원 위의 특정 점으로 translation 시킨다. 그리고 MC를 시계 반대방향으로 원 위에서 이동시킨다.

한편 MC가 WC 위의 한 원 위를 이동하는 속도와 같은 속도로, 원점을 기준 반시계 방향으로 회전하는 직선이 있다. 이 때 이 직선을 기준으로 MC 좌표계의 모든 점, 즉 MC 좌표계에서 두 개의 cocktail object를 대칭시킨다.

따라서 새로운 Modeling 좌표계 위를 움직이는 한 개의 cocktail object를 MC내의 원점 기준 대칭시키고, 이러한 MC 좌표계의 모든 점을 World 좌표계 위에서 radius 를 반지름으로 하는 원 위의 한 점으로 이동시킨 후 원 위를 움직이는 translation formation을 적용시키고, 이 MC가 원을 움직이는 속도와 같은 속도로 움직이는 한 직선을 기준으로 MC를 대칭시키는 transformation을 구현하였다.

‘l’ 또는 ‘L’ 버튼을 누르면 WC위에 그려진 MC좌표계 및 원점을 지나는 직선과, 이 직선에 대칭된 MC를 확인할 수 있다.



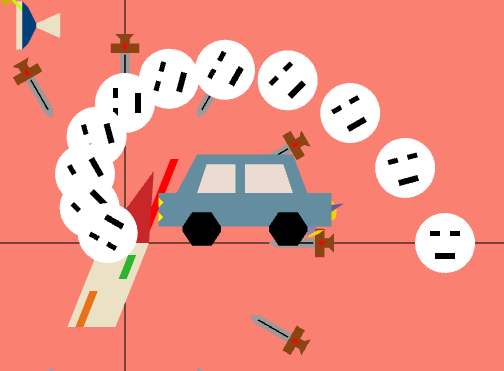
< 초록색으로 표현된 직선과, 두 개의 Modeling 좌표계 >

1. **FACE** – rotation, translation

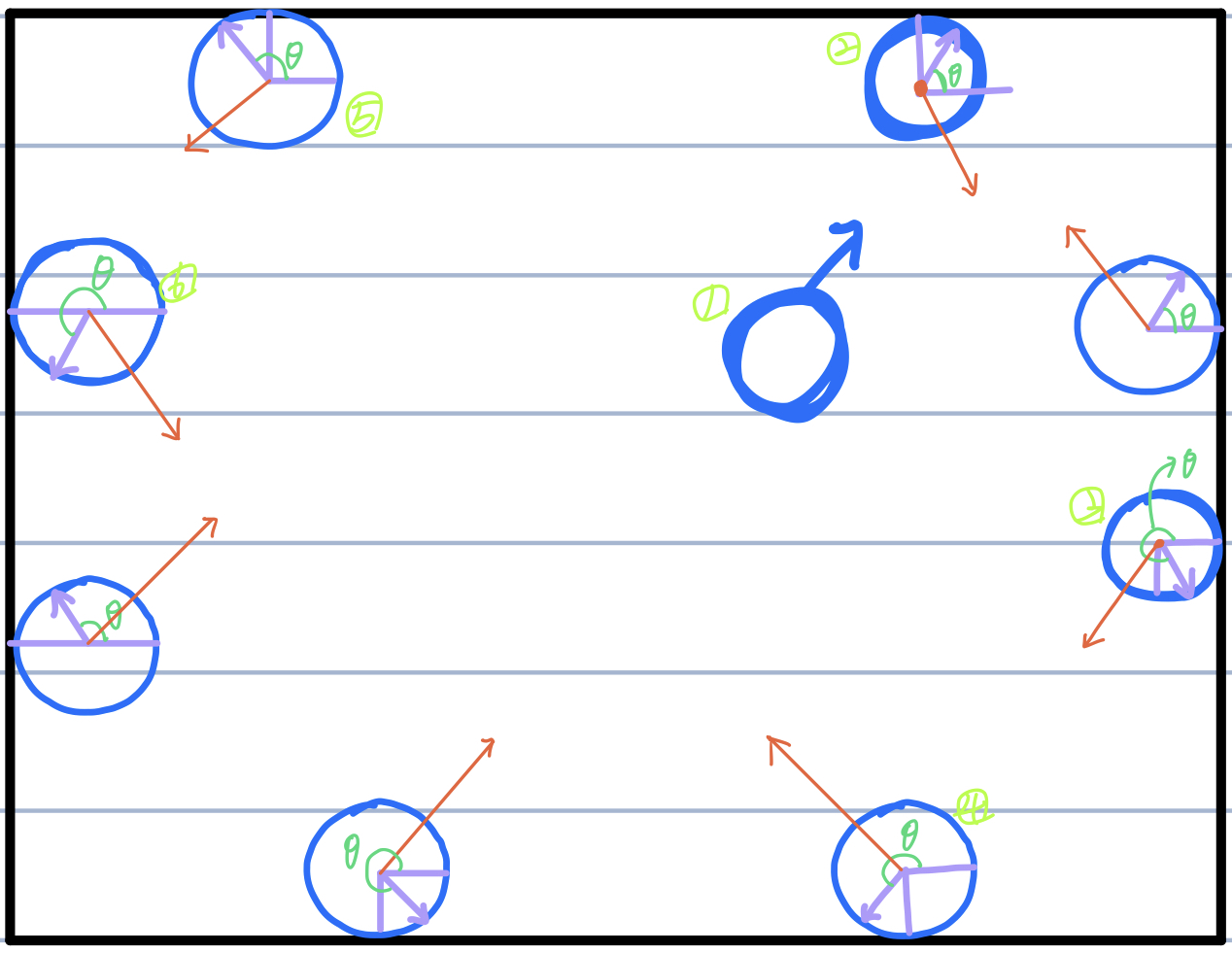


위와 같은 모습으로 직접 모델링한 물체이다.(왼쪽 선분 제외) 얼굴의 원은 360개의 삼각형으로 이루어졌으며, 눈과 입은 각각 4개의 좌표를 이용해 그림을 그렸다.

해당 object는 키보드 ‘f’ 또는 ‘F’ 키를 누르면 생성된다. 각 face는 일정한 시간 간격에서 원점 기준 15 degree 간격을 두고 순차적으로 생성된다.



위와 같은 형태로 360 degree 총 24개의 object가 생성되면, 각 object들은 각각 좌표계 위에서 임의로 결정된 한 직선 위를 움직인다. 이동 중 벽(window 창의 4면)을 만나면 벽에 부딪히고 경로가 달라진다. 이 때 벽에 부딪히는 경우를 아래와 같이 구분하여 벽에서 튕겨 나오는 경로를 자연스럽게 구현했다.



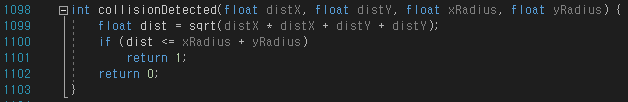
1. **ANGRY FACE** – translation



위와 같은 모습으로 모델링한 object이다. 이 물체를 화면에 그리는 크기는 WINDOW 창의 높이만큼이다.

‘a’ 또는 ‘A’ 버튼을 누르면 위의 object가 화면의 가장 왼쪽에서 오른쪽으로 이동하며 화면 밖으로 사라진다. 간단한 translation 변환만 수행하지만, object가 이동하며 6번의 face object들과 collision detect function을 수행한다.

ANGRY FACE object가 이동하며 물체의 오른과 FACE object가 충돌하면, 각각 충돌한 FACE object들은 ANGRY FACE object로부터 튕겨져 나온다. 이러한 collision detect를 수행하는 함수는 아래와 같다.



함수의 parameter로 사용된 distX와 distY는 충돌이 일어났는지 확인할 두 물체의 중심이 각각 x축 방향과 y축 방향으로 떨어진 거리를 의미한다.(부호 무시) xRadius와 yRadius는 각각 두 물체의 반지름을 나타낸다. 따라서 두 물체의 중심 사이의 거리(dist)가 두 물체의 반지름을 더한 값보다 작거나 같으면 collision이 일어난 것이고, 그렇지 않으면 일어나지 않은 것으로 구현하였다.

한편 ANGRY FACE object의 중심이 x축 양의 방향에서 window width / 10 보다 큰 경우, FACE object는 오른쪽 화면에서 왼쪽화면으로 넘어가도록 구현하였다.