과목 명: 그래픽스

담당 교수 명: 임 인 성

<<Assignment 3>>

**서강대학교 컴퓨터학과**

**[20151616]**

**[최승환]**

목 차

1. 프로그램 개요 3

2. 프로그램 설명 3

2.1 프로그램 사용법 3

기능 설명 3

3. 물체의 배치 및 움직임 구현 3

3.1 움직이는 호랑이에 모델링 변환 적용

3.2 계층적 모델링 방식의 자동차 3

3.3 월드에 배치한 5개의 물체

3.4 키보드나 마우스를 통해 조작할 수 있는 물체의 움직임

4. 카메라의 배치 및 움직임 구현 3

4.1 주 카메라

4.2 부 카메라

4.3 움직이는 자동차 운전석에 고정된 카메라

4.4 움직이는 호랑이 눈에 고정된 카메라

5 추가 구성 : 거미줄, 자동차 물리 4

5.1 거미줄 4

5.2 자동차의 움직임 4

6. 코드 내 적용한 물리 추가 설명 4

7. 참조 링크

# 프로그램 개요

OpenGL 을 사용해 작성한 그래픽스 3차 과제에 대한 코드입니다. 3차원 공간 내에 물체들을 배치하고, 특정 키를 통해 물체의 움직임을 조작할 수 있습니다. 카메라의 각도를 조작하거나, 특정 위치에서 카메라를 볼 수 있도록 만들어져 있습니다.

# 프로그램 설명

## 프로그램 사용법

화면 내에는 자동으로 움직이는 물체들과 조작을 통해 움직일 수 있는 물체들로 나뉘어져 있습니다. 사용자 입력에 의해 작동되는 키는 다음과 같습니다.

a - driver 기준의 디스플레이 모드로 전환합니다.

s - tiger 기준의 디스플레이 모드로 전환합니다.

d - driver 기준의 뷰포트를 화면 오른쪽 하단에 추가합니다. (월드뷰에서만 가능합니다.)

t - tiger 기준의 뷰포트를 화면 오른쪽 상단에 추가합니다. (월드뷰에서만 가능합니다.)

w - world 기준의 뷰포트(기본화면) 상태로 되돌립니다.

p - teapot 을 움직이도록 만듭니다. 만약 움직이고 있었다면, 그 상태로 멈추게 만듭니다.

b - box 를 움직이도록 만듭니다. 만약 움직이고 있었다면 그 상태로 멈추게 만듭니다.

c - tiger 를 움직이도록 만듭니다. 만약 움직이고 있었다면, 멈추고, 출발점으로 되돌립니다.

**사용 시 주의사항**

* **d, t 키는 월드뷰에서만 사용가능합니다. 다른 뷰에서 사용할 시 아무 일도 일어나지 않습니다.**
* **driver는 궤도 여행 중 한동안 아무 물건도 배치되지 않는 공간을 날아갑니다. 버그나 화면이 꺼진 것이 아니니, 기다리면 다시 물건들이 보이는 궤도를 여행합니다.**
* **좌, 우 방향키를 통한 시점 변경은 상, 하 방향키를 누를 시 중앙으로 초기화됩니다.  
  (카메라의 벡터가 변하거나, 지정된 지점을 초과하는 경우를 막기 위한 설정입니다.)**

# 기능 설명

## 움직이는 호랑이에 모델링 변환 적용

### 기능

12개의 프레임을 가지고 있는 호랑이 모델이 월드에 배치되어 있습니다. 이 호랑이는 프레임별로 모션을 행하고, ‘c’ 키를 누를 시 제자리에서 y축 방향으로 탄성운동을 시작합니다. 이 모델에는 카메라를 설치할 수 있습니다. 해당 내용은 뒤에서 다루도록 하겠습니다.

* + - 1. 호랑이는 translate, rotate scale 등을 통해 월드 좌표에 배치되어 있습니다. 이 좌표가 변하는 건 c 키를 눌렀을 때 입니다.

c 키를 누를 시 crazycow\_flag 가 1로 설정됩니다. 만약 원래 1이었다면, 0으로 설정됩니다.(토글 가능)

crazycow\_flag가 1로 설정되어 있을 시, display callback 함수에서는 jump\_crazycow() 를 호출합니다. jump\_crazycow 는 전역변수로 설정된 속도를 프레임별로 서서히 줄이고, 높이가 0이 되면 속도를 초기화시켜주는 방식으로 만들어져 있습니다. (탄성운동)

이후 이 좌표들을 이용해, 월드에 배치시킵니다.

코드부

(초기 설계에서는 모델을 소로 해서 변수명이 tiger가 아닌 점 양해 부탁드립니다.)

void switch\_crazycow()

{

if (!crazycow\_flag)

{

crazycow\_flag = 1;

}

else

{

crazycow\_flag = 0;

crazycow\_coord = -40.0f;

crazycow\_height = 0.001f;

crazycow\_speed = 0.5f;

}

}

void jump\_crazycow()

{

if (crazycow\_height < 0.0f)

{

crazycow\_speed = -1 \* crazycow\_speed;

crazycow\_height = 0.001f;

return;

}

crazycow\_speed = crazycow\_speed - 0.01f;

crazycow\_coord = crazycow\_coord + 0.05f;

crazycow\_height = crazycow\_height + crazycow\_speed;

}

## 계층적 모델링 방식의 자동차

### 기능

자동차는 프로그램이 시작되자마자 자동으로 특정 궤도를 따라 운동합니다. 자동차와 그 바퀴, 너트는 각각의 위치와 회전한 각도를 계층적으로 가지고 있으며, 자동차가 움직인 거리, 자동차의 진행방향까지 고려해, 자동차의 몸체와 바퀴의 상태를 조정합니다.

* + - 1. 자동차의 바퀴는 자동차의 몸체에, 바퀴의 너트는 자동차 바퀴에 소속되어 움직입니다.  
          (자동차의 좌표에 관한 설명은 복잡하므로 후반부에 하도록 하겠습니다.)

코드부

void move\_car()

{

car\_rotation = car\_rotation + 0.2f;

if (car\_rotation > 360.0f)

{

car\_rotation = 0.0f;

}

car\_x = CAR\_ROUTE\_RADIUS \* -sin(car\_rotation\*TO\_RADIAN);

car\_y = (CAR\_ROUTE\_RADIUS \* -cos(car\_rotation\*TO\_RADIAN)) + CAR\_ROUTE\_RADIUS;

if (car\_twist > 45.0f)

{

car\_z\_flag = 0;

}

else if (car\_twist < -45.0f)

{

car\_z\_flag = 1;

}

if (car\_z\_flag)

{

car\_twist += 0.4f;

car\_z = car\_z + sin(car\_twist\*TO\_RADIAN);

}

else

{

car\_twist -= 0.4f;

car\_z = car\_z + sin(car\_twist\*TO\_RADIAN);

}

}

바퀴에 들어가는 코드 (바퀴의 각도와 좌표 추후 설명)

ModelMatrix\_CAR\_WHEEL = glm::translate(ModelMatrix\_CAR\_BODY, glm::vec3(-3.9f, -3.5f, 4.5f));

ModelMatrix\_CAR\_WHEEL = glm::rotate(ModelMatrix\_CAR\_WHEEL, car\_twist\* TO\_RADIAN, glm::vec3(0.0f, 1.0f, 0.0f));

ModelMatrix\_CAR\_WHEEL = glm::rotate(ModelMatrix\_CAR\_WHEEL, car\_rotation \* CAR\_ROUTE\_RADIUS / 3.4f \* TO\_RADIAN ,glm::vec3(0.0f, 0.0f, 1.0f));

ModelViewProjectionMatrix = ViewProjectionMatrix \* ModelMatrix\_CAR\_WHEEL;

glUniformMatrix4fv(loc\_ModelViewProjectionMatrix, 1, GL\_FALSE, &ModelViewProjectionMatrix[0][0]);

draw\_wheel\_and\_nut(); // draw wheel 0

## 월드에 배치한 5개의 물체들

### 기능

### 월드에는 호랑이와 자동차를 제외하고 5종류의 물체가 존재합니다. 하나는 정적인 물체, 월드의 물체 중 세 개는 유저의 입력에 따라 운동하며, 1종류는 16프레임의 모션을 가지고 있는 거미, 나머지 하나는 텍스트 파일을 작성해 제가 만든 물체입니다.

* + - 1. 소 3형제  
          소 3형제는 시간에 따라 색이 연속적으로 바뀝니다. 프레임별로 호출되는 디스플레이 함수에서 색에 대한 변수를 계속 늘리거나 줄이면, 조금씩 색이 변하게 됩니다.

cow\_gradation = cow\_gradation + 0.005f;

if (cow\_gradation > 1.0f)

{

cow\_gradation = 0.100f;

}

glUniform3f(loc\_primitive\_color, 0.705f, cow\_gradation, 0.314f);

* + - 1. 박스  
          박스는 유저의 입력에 의해 운동하는 물체입니다. b 키를 누르면 박스의 운동이 시작되고, b키를 다시 누르면 그 자리에서 멈춥니다. 운동하는 동안은 색이 변합니다.  
           
          박스가 하는 운동은 진동운동입니다. 속도를 점차 늦춰서 -가 되면, 물체가 뒤로 움직이기 시작하는데, 뒤로 움직이다 보면 출발점을 지나게 됩니다. 출발점을 지날 때부터 반대방향으로 가속하기 시작합니다. 이렇게 하면, 같은 궤도를 반복하며 가속운동을 하는 물체를 얻을 수 있습니다.  
         (ex. 바이킹)

하지만, 이 운동은 사인 그래프로도 재연할 수 있습니다. 이 때문에, 진동운동하는 물체를 만들 때, 하나의 변수, 각도만을 이용해서 그 축의 좌표를 구해낼 수 있습니다.

void pendulum\_box()

{

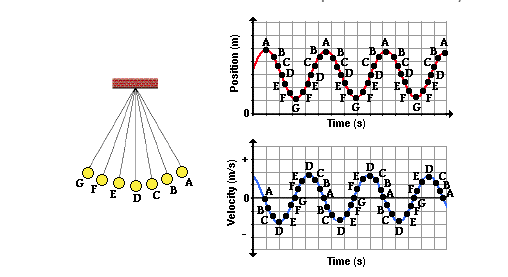
rotation\_angle\_box = rotation\_angle\_box + 0.15;

}

...

ModelMatrix\_BOX = glm::translate(ModelMatrix\_BOX, glm::vec3(10 \* sin(rotation\_angle\_box), 1.0f, -20.0f));

…  
  
진동운동은 진자운동의 x축 좌표와 같은 운동을 합니다. sin 함수를 사용하면 진자운동의 x 축 좌표를 얻을 수 있습니다.



* + - 1. 찻잔  
          찻잔은 p 키를 누르면 움직이는 물체입니다. 박스와 마찬가지로 토글 가능하며, 색이 변하면서 움직입니다. 찻잔은 원점을 기준으로 특정 거리를 두고 원 궤도 운동을 합니다. 찻잔을 원래 각도로 유지하면서, 회전시키려면, rotate 이후에 translate 를 사용하면 되지만, 반대의 순서로 사용을 해, 회전 각도가 불규칙한 위성운동과 같은 형태를 만들었습니다.

(코드상 순서는 rotate -> translate로 보이지만, 실제로는 반대)

...  
ModelMatrix\_TEAPOT = glm::rotate(glm::mat4(1.0f), -rotation\_angle\_teapot \* TO\_RADIAN, glm::vec3(0.0f, 1.0f, 1.0f));

ModelMatrix\_TEAPOT = glm::translate(ModelMatrix\_TEAPOT, glm::vec3(-20.0f, 1.0f, 10.0f));

…

* + - 1. 거미  
          거미는 시간에 따라 프레임을 달리하는, 모션이 있는 물체입니다. 이를 조정하기 위해 timer\_scene 함수에서 현재 프레임을 받아오도록 cur\_frame\_spider 에 값을 저장합니다.  
            
         void timer\_scene(int timestamp\_scene) {

rotation\_angle\_car = (timestamp\_scene % 360)\*TO\_RADIAN;

cur\_frame\_spider = timestamp\_scene % N\_SPIDER\_FRAMES;

cur\_frame\_tiger = timestamp\_scene/10 % N\_TIGER\_FRAMES;

glutPostRedisplay();

glutTimerFunc(10, timer\_scene, (timestamp\_scene + 1) % INT\_MAX);

}

단순히 거미가 프레임별로 움직이는 데에만, prepare\_spider , timer\_scene, draw\_spider 세 개의 함수가 사용됩니다. 이 코드는 양이 상당해 보고서에는 생략하겠습니다.

void rotate\_spider()

{

if (rotation\_angle\_spider > 360.0f\*TO\_RADIAN)

{

rotation\_spider\_flag = 0;

scale\_spider = -1;

}

else if(rotation\_angle\_spider < 0.0f)

{

rotation\_spider\_flag = 1;

scale\_spider = 1;

}

if (rotation\_spider\_flag)

{

rotation\_angle\_spider = rotation\_angle\_spider + 0.01f;

}

else

{

rotation\_angle\_spider = rotation\_angle\_spider - 0.01f;

}

}

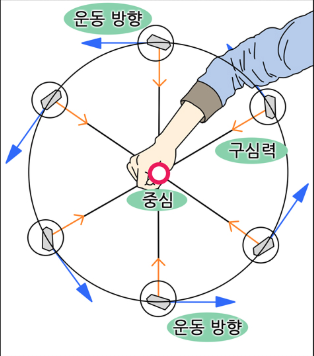
거미는 한 바퀴를 돌고 난 후에는 반대 방향으로 돕니다. 이를 위해, 현재 돌고 있는 각도를 저장하고, 그 각도가 360도를 넘으면, 반대 방향으로 돌도록 합니다. 이 때 거미는 항상 중심을 오른쪽에 두고 돌아야 하기 때문에, 현재 돌고 있는 각도만큼 거미 자체를 돌린 후, 원의 좌표에 맞게 위치시켜야 합니다.  
  
하지만, 반지름이 일정하다는 점을 이용하면, 반지름만큼 이동시킨 후에 회전시키는 방식을 사용할 수도 있습니다. (찻잔을 회전시킨 방식에서 조금 변형)  
ModelMatrix\_spider = glm::translate(glm::mat4(1.0f), glm::vec3(WEB\_START\*1.55f, 0.0f, 0.0f));

ModelMatrix\_spider = glm::rotate(ModelMatrix\_spider, 90.0f\*TO\_RADIAN, glm::vec3(0.0f, 1.0f, 0.0f));

ModelMatrix\_spider = glm::rotate(ModelMatrix\_spider, rotation\_angle\_spider, glm::vec3(0.0f, 0.0f, 1.0f));

ModelMatrix\_spider = glm::translate(ModelMatrix\_spider, glm::vec3(20.0f, 0.0f, 10.0f));

...



* + - 1. 거미줄  
           
         거미줄은 제가 작성한 web.txt. 파일을 읽어와 그리는 물체입니다. 월드 상에서 거미줄은 굉장히 많은 숫자의 점을 가지고 있지만, 점들 간의 규칙성이 간단하기 때문에, 12개의 점만을 사용해 만들 수 있습니다.   
         이 12개의 점을 반복문을 통해 멀어질 수록 더 크게 그리도록 만들어 하나의 거미줄을 만들 수 있습니다.  
         그리고 마지막으로 이 거미줄 함수를 반대로 그리도록 해 대칭적인 구조를 얻을 수 있습니다.

int read\_web\_file(GLfloat \*\*object, char \*filename) {

int i, j, n\_vertices;

float \*flt\_ptr;

float x, y, z;

float x\_screw = 1.0f;

float web\_scale = 1.0f;

FILE \*fp;

fprintf(stdout, "Reading path from the path file %s...\n", filename);

fp = fopen(filename, "r");

if (fp == NULL) {

fprintf(stderr, "Cannot open the path file %s ...", filename);

return -1;

}

fscanf(fp, "%d", &n\_vertices);

\*object = (float \*)malloc(n\_vertices \* 3 \* sizeof(float) \* WEB\_DEPTH);

if (\*object == NULL) {

fprintf(stderr, "Cannot allocate memory for the path file %s ...", filename);

return -1;

}

flt\_ptr = \*object;

for (i = 0; i < n\_vertices; i++) {

fscanf(fp, "%f %f %f", &x, &y, &z);

web\_scale = 1.0f;

x += WEB\_START;

for (j = 0; j < WEB\_DEPTH; ++j)

{

\*(flt\_ptr + j\*n\_vertices \* 3) = x\*web\_scale+j\*x\_screw ;

\*(flt\_ptr + j\*n\_vertices \* 3 + 1) = y\*web\_scale;

\*(flt\_ptr + j\*n\_vertices \* 3 + 2) = z\*web\_scale;

web\_scale = web\_scale \* WEB\_RATIO;

}

flt\_ptr += 3;

}

fclose(fp);

fprintf(stdout, "Read %d vertices successfully.\n\n", n\_vertices);

return WEB\_DEPTH\*n\_vertices;

}

대칭적인 구조를 위한 코드  
  
ModelViewProjectionMatrix = ViewProjectionMatrix \* ModelMatrix\_web;

glUniformMatrix4fv(loc\_ModelViewProjectionMatrix, 1, GL\_FALSE, &ModelViewProjectionMatrix[0][0]);

draw\_web();

ModelMatrix\_web = glm::scale(glm::mat4(1.0f), glm::vec3(-1.0f, 1.0f, 1.0f));

ModelViewProjectionMatrix = ViewProjectionMatrix \* ModelMatrix\_web;

glUniformMatrix4fv(loc\_ModelViewProjectionMatrix, 1, GL\_FALSE, &ModelViewProjectionMatrix[0][0]);

draw\_web();

## 키보드나 마우스를 통해 조작할 수 있는 물체의 움직임

### 기능 p - teapot 을 움직이도록 만듭니다. 만약 움직이고 있었다면, 그 상태로 멈추게 만듭니다.

b - box 를 움직이도록 만듭니다. 만약 움직이고 있었다면 그 상태로 멈추게 만듭니다.

c - tiger 를 움직이도록 만듭니다. 만약 움직이고 있었다면, 멈추고, 출발점으로 되돌립니다.  
  
다음 3개의 키를 이용해 물체의 운동을 조절할 수 있습니다. 이 중 p, b 는 토글이 가능하고, c는 누를 때마다 원점으로 돌아간 후, 다시 누르면 다시 시작합니다.

1. **카메라의 배치 및 움직임 구현**
   1. 주 카메라
      1. 기능  
          이 프로그램에서 지원하는 시점은 세가지이다. 월드뷰, 운전자, 호랑이 기준으로 되어 있다. 주 카메라를 바꾸는 건   
         a - driver 기준의 디스플레이 모드로 전환합니다.

s - tiger 기준의 디스플레이 모드로 전환합니다.

w - world 기준의 뷰포트(기본화면) 상태로 되돌립니다.  
위 세 개의 키를 사용합니다. 각각의 키를 누르면, 해당 시점의 카메라를 사용할 수 있습니다.

A.프로그램이 시작되면 월드뷰 기준으로 적당한 거리에서 원점을 바라보고 있습니다.   
B.SHIFT 키를 누르고 드래그하면, 좌 우로 움직인 정도에 따라 줌인/줌아웃이 가능합니다.

C.월드뷰에 대해서, 좌 우 키를 누르면 해당 방향으로 30도씩 회전한 극좌표로 카메라의 위치를 옮기며, 바라보는 방향은 원점으로 같습니다. 상 하 키에 대해서는 z축 기준으로 + , - 방향(수직 방향) 으로 볼 수 있습니다. 하지만, 이 경우에는 좌 우 키를 사용해 움직였던 각도가 0도로 초기화됩니다.

(좌->상->우->하 같은 순서를 밟으면, 카메라가 다른 방향으로 회전되는 것을 막기 위함)

다른 뷰 기준으로도 월드뷰와 같이 카메라 조작이 가능하도록, camera\_wv.pos 를 일부 사용했습니다.

void set\_ViewMatrix\_for\_driver(void) {

glm::mat4 Matrix\_CAMERA\_driver\_inverse, Matrix\_CAMERA\_rotation;

Matrix\_CAMERA\_rotation = glm::mat4(camera\_dv.uaxis.x, camera\_dv.vaxis.x, camera\_dv.naxis.x, 0.0f,

camera\_dv.uaxis.y, camera\_dv.vaxis.y, camera\_dv.naxis.y, 0.0f,

camera\_dv.uaxis.z, camera\_dv.vaxis.z, camera\_dv.naxis.z, 0.0f,

0.0f, 0.0f, 0.0f, 1.0f);

Matrix\_CAMERA\_driver\_inverse = ModelMatrix\_CAR\_BODY \* ModelMatrix\_CAR\_BODY\_to\_DRIVER;

ViewMatrix = glm::affineInverse(Matrix\_CAMERA\_driver\_inverse);

ViewMatrix = glm::translate(ViewMatrix, -camera\_wv.pos);

ViewMatrix = Matrix\_CAMERA\_rotation \* ViewMatrix;

ViewProjectionMatrix = ProjectionMatrix \* ViewMatrix;

}

(월드뷰와 호랑이 시점의 코드는 양이 많아 생략하겠습니다.)

주의! : 호랑이 시점의 뷰는 호랑이가 앞을 바라보는 부분이 아닌, 호랑이의 옆 부분으로 바라보도록 작성되어 있습니다. 이는 호랑이의 위치 상 앞을 바라봤을 때 보이는 물체가 거의 없기 때문이니, 양해 부탁드립니다.

* 1. 부 카메라

1. 부 카메라는 두 개의 키를 이용해 조작할 수 있습니다.  
   d - driver 기준의 뷰포트를 화면 오른쪽 하단에 추가합니다. (월드뷰에서만 가능합니다.)

t - tiger 기준의 뷰포트를 화면 오른쪽 상단에 추가합니다. (월드뷰에서만 가능합니다.)  
  
해당 키는 누를 때마다 토글이 되고, 주 카메라에서 조작하는 것과 같은 방식으로 세상이 도시됩니다. d나 t를 눌러서 뷰포트를 추가하면, 총 3개까지 뷰포트가 추가되고, 추가된 뷰포트가 없을 경우에는 다시 월드뷰와 같은 크기로 돌아옵니다.

1. 이 카메라는 월드뷰와 동일한 방식으로 SHIFT 마우스 좌클릭 드래그를 통해 줌인/ 줌아웃이 가능하고, 상 하 좌 우 방향키를 통해 시점도 월드뷰와 같은 방식으로 바꿀 수 있습니다.
2. 카메라는 키보드를 사용해 월드뷰와 같은 방식으로 보는 방향을 바꿀 수 있습니다.
3. 카메라는 마우스를 이용해 줌인/줌아웃이 월드뷰와 같이 가능합니다.  
     
     
   if ((camera\_driver\_flag == 0) && (camera\_tiger\_flag == 0))

{

glViewport(0, 0, window\_width, window\_height);

set\_ViewMatrix\_for\_world\_viewer();

draw\_objects\_in\_world();

}

else

{

glViewport(0, 0, window\_width / 4 \* 3, window\_height);

set\_ViewMatrix\_for\_world\_viewer();

draw\_objects\_in\_world();

if (camera\_driver\_flag)

{

glViewport(window\_width / 4 \* 3 + 1, 0, window\_width / 4, window\_height / 4); set\_ViewMatrix\_for\_driver();

draw\_objects\_in\_world();

}

if (camera\_tiger\_flag)

{

glViewport(window\_width / 4 \* 3 + 1, window\_height / 2, window\_width / 4, window\_height / 4);

set\_ViewMatrix\_for\_tiger();

draw\_objects\_in\_world();

}

}

* 1. 움직이는 자동차 운전석에 고정된 카메라  
     (4.1에서 설명한 기능의 일부이므로 생략하겠습니다.)
  2. 움직이는 호랑이 눈에 고정된 카메라

(4.1에서 설명한 기능의 일부이므로 생략하겠습니다.)

5. 추가구성

거미줄

거미줄을 구성하는 점은 굉장히 많지만, 거미줄을 만드는데 참조한 파일에는 12개의 점 뿐입니다. 이는 거미줄이 반복적인 구조로 이루어졌기 때문입니다. 거미줄은 중심을 기준으로 다각형이 반복적으로 커지면서 그려나가는 구조입니다. 이를 재현하기 위해서 다음과 같은 상수들을 선언했습니다.

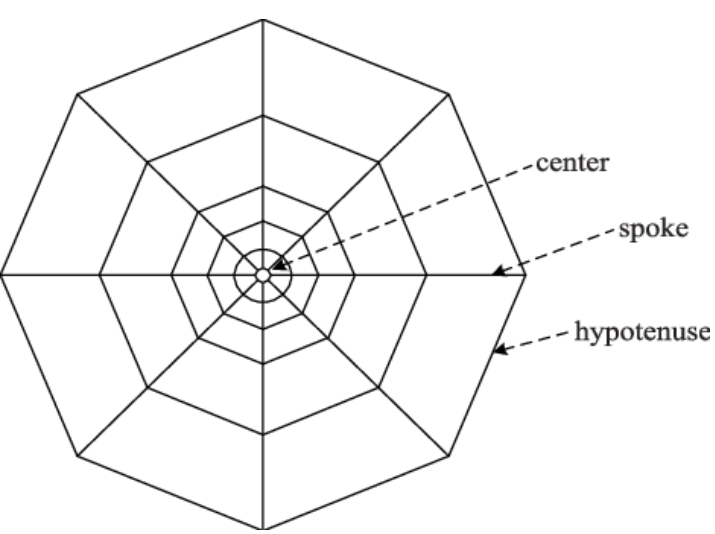
#define WEB\_SIZE 7.5f

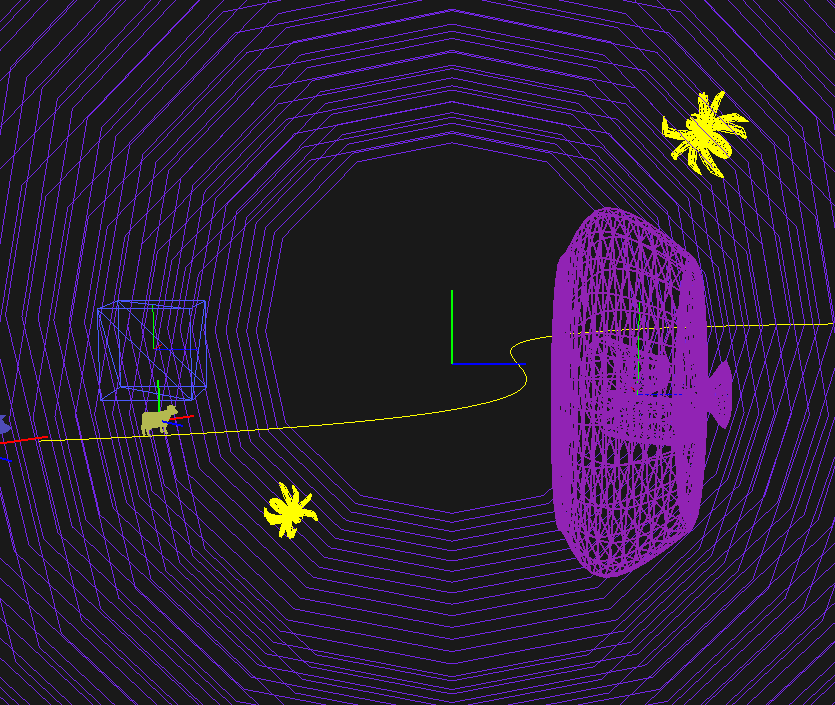
#define WEB\_RATIO 1.05f

#define WEB\_DEPTH 50

#define WEB\_CYCLE 13

#define WEB\_START -20.0f





WEB\_SIZE 는 첫 한 바퀴의 크기 비율을 의미하고 RATIO 는 한 바퀴의 거미줄이 한 번에 커지는 비율을 의미합니다. 두번째바퀴는 1.05배 세번째바퀴는 1.05^2 배 같은 방식으로 점점 커져나갑니다.WEB\_START 는 거미줄의 중심점의 x 좌표를 의미하고, DEPTH 는 바퀴 수를 의미합니다.

for (i = 0; i < n\_vertices; i++) {

fscanf(fp, "%f %f %f", &x, &y, &z);

web\_scale = 1.0f;

x += WEB\_START;

for (j = 0; j < WEB\_DEPTH; ++j)

{

\*(flt\_ptr + j\*n\_vertices \* 3) = x\*web\_scale+j\*x\_screw ;

\*(flt\_ptr + j\*n\_vertices \* 3 + 1) = y\*web\_scale;

\*(flt\_ptr + j\*n\_vertices \* 3 + 2) = z\*web\_scale;

web\_scale = web\_scale \* WEB\_RATIO;

}

flt\_ptr += 3;

}

위 코드의 경우 하나의 점을 버퍼에 저장할 때마다, 포인터가 3씩 밀려나게 됩니다. 이를 고려하면서, DEPTH 에 비례하는 수의 바퀴를 버퍼에 저장합니다.

...

ModelViewProjectionMatrix = ViewProjectionMatrix \* ModelMatrix\_web;

glUniformMatrix4fv(loc\_ModelViewProjectionMatrix, 1, GL\_FALSE, &ModelViewProjectionMatrix[0][0]);

draw\_web();

ModelMatrix\_web = glm::scale(glm::mat4(1.0f), glm::vec3(-1.0f, 1.0f, 1.0f));

ModelViewProjectionMatrix = ViewProjectionMatrix \* ModelMatrix\_web;

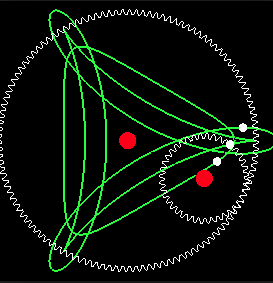
glUniformMatrix4fv(loc\_ModelViewProjectionMatrix, 1, GL\_FALSE, &ModelViewProjectionMatrix[0][0]);

draw\_web();

…

이후에 prepare\_web() 과 draw\_web() 을 이용해, 저장되어 있는 거미줄을 그릴 수 있습니다. 하지만, 이 프로젝트의 경우 원점을 기준으로 대칭인 두 개의 web을 그렸는데, 이는 단순히 x 축 -1 배 scale 을 이용해 그릴 수 있었습니다.

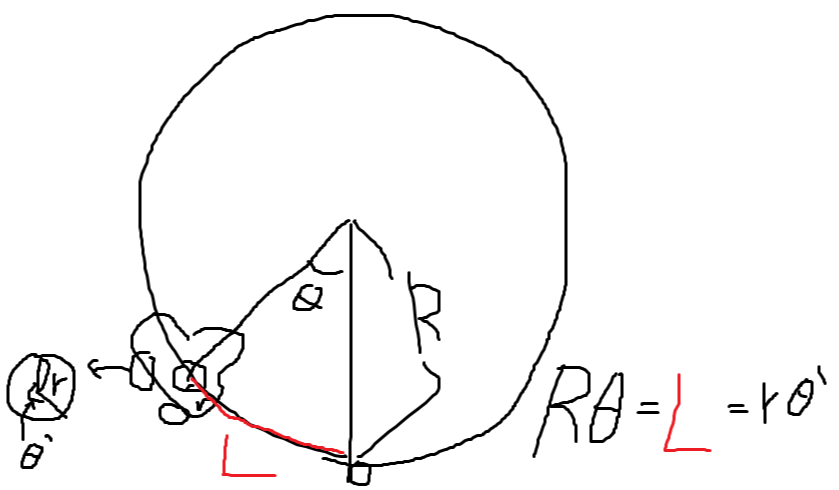
자동차 바퀴



이 프로젝트에서 자동차는 원 궤도를 따라 위성과 같은 운동을 합니다. 위 그림에서 빨간 색 중심점이 차의 구심점, 바깥쪽의 빨간 점이 차의 위치와 같습니다. 이 경우, 자동차의 바퀴를 어떻게 조작해야 가장 현실과 유사하게 조작할 수 있을까? 라는 물음에서 시작되었습니다.

자동차 바퀴마다 중심으로부터 조금씩 다른 위치에 존재하지만, 자동차의 구심원 반지름이 꽤 크다는 것을 감안했을 때는 바퀴 각각의 위치 차이는 미미합니다. 이를 근사해서 생각하면, 자동차의 중심점에 바퀴가 있다고 가정하고, 그 회전 정도를 구할 수 있습니다.

위 구조는 큰 원 내에서 작은 원이 굴러가는 것과 같습니다. 이를 Cycloid 라고 합니다. 이 프로젝트에서 바퀴가 회전하는 정도는 Cycloid에서 작은 원이 회전하는 정도와 같습니다. 차의 위치를 구현하기 위해서 차가 원점에서 몇도만큼 회전한 구심원 위의 점에 있는지를 우리는 알고 있습니다.



따라서 theta 를 통해 theta ‘ 를 구할 수 있습니다. theta ‘ = theta \* R / r 이 될 것입니다. 바퀴의 반지름은 3.3이 조금 넘으므로 3.4로 잡고 계산식을 구하면 다음과 같은 코드가 나옵니다.

ModelMatrix\_CAR\_WHEEL = glm::rotate(ModelMatrix\_CAR\_WHEEL, car\_rotation \* CAR\_ROUTE\_RADIUS / 3.4f \* TO\_RADIAN ,glm::vec3(0.0f, 0.0f, 1.0f));

그리고 마지막으로 굴러가고 있는 바퀴가 방향전환을 자연스럽게 하기 위한 코드가 필요합니다.

ModelMatrix\_CAR\_WHEEL = glm::rotate(ModelMatrix\_CAR\_WHEEL, car\_twist\* TO\_RADIAN, glm::vec3(0.0f, 1.0f, 0.0f));

이는 car\_twist 라는 변수를 저장해 놓고 사용했습니다. car\_twist 는 현재 자동차가 바라보는 방향입니다. 이 방향은 플래그를 사용해 다음과 같이 작성했습니다.

void move\_car()

{

car\_rotation = car\_rotation + 0.2f;

if (car\_rotation > 360.0f)

{

car\_rotation = 0.0f;

}

car\_x = CAR\_ROUTE\_RADIUS \* -sin(car\_rotation\*TO\_RADIAN);

car\_y = (CAR\_ROUTE\_RADIUS \* -cos(car\_rotation\*TO\_RADIAN)) + CAR\_ROUTE\_RADIUS;

if (car\_twist > 45.0f)

{

car\_z\_flag = 0;

}

else if (car\_twist < -45.0f)

{

car\_z\_flag = 1;

}

if (car\_z\_flag)

{

car\_twist += 0.4f;

car\_z = car\_z + sin(car\_twist\*TO\_RADIAN);

}

else

{

car\_twist -= 0.4f;

car\_z = car\_z + sin(car\_twist\*TO\_RADIAN);

}

}

7. 참조 링크

박스의 진동운동과 사인의 연관성

<https://en.wikipedia.org/wiki/Sine_wave>

원 궤도를 운동하는 자동차 바퀴,사이클로이드

<https://ko.wikipedia.org/wiki/%EC%82%AC%EC%9D%B4%ED%81%B4%EB%A1%9C%EC%9D%B4%EB%93%9C>