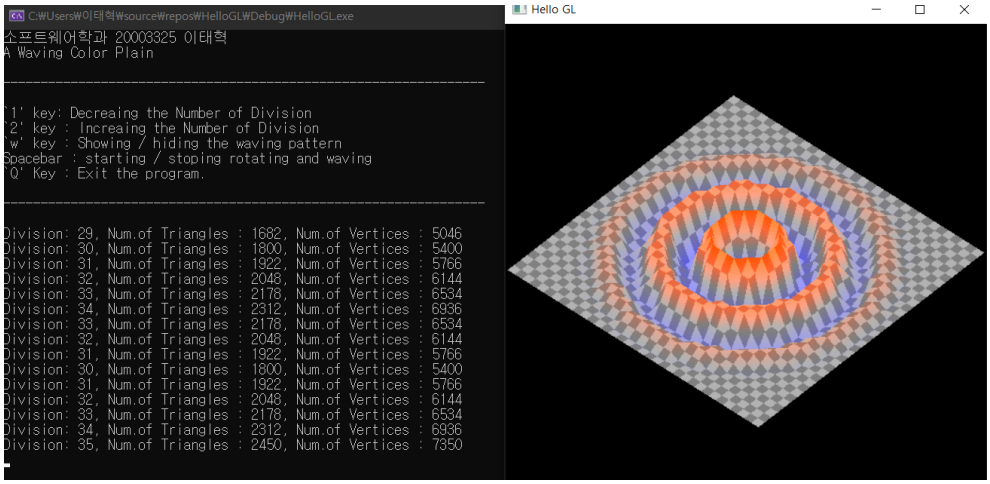
컴퓨터그래픽스 과제 1

소프트웨어학과 20003325 이태혁

실행결과



풀이과정

1. 큰 plain 하나 생성

```
Myplain plain;
```

Myplain 클래스를 생성하고 plain객체 하나를 만들었다. plain의 가로, 세로 길이는 1.4로 설정했다.

2. plain의 각도를 비스듬하게 눕힌 후 Z축 기준으로 회전

```
float angleX = -45;
float angleZ = uTime * 200; // 초당 몇 도씩 돌게 만듦

float radX = angleX / 180.0f * 3.141592f;
float radZ = angleZ / 180.0f * 3.141592f;

// -------------------- 회전행렬 ------------------------

  rotX[1][1] = cos(radX);   rotX[2][1] = -sin(radX);
  rotX[1][2] = sin(radX);   rotX[2][2] = cos(radX);

  rotZ[0][0] = cos(radZ);   rotZ[1][0] = -sin(radZ);
  rotZ[0][1] = sin(radZ);   rotZ[1][1] = cos(radZ);
```

X축, Z축 회전행렬을 생성하여 평면에 적용시켰다. X축을 기준으로 -45도 정도 돌리면 z값이 양인 부분이 위쪽을 향하도록 돌아간다. 그리고 시간이 지날수록 값이 커지는 변수 uTime을 만들어서 uniform 타입으로 shader에 넘겨주고, uTime을 이용하여 Z축을 기준으로 계속 회전하도록 구현했다.
이 때 주의할 점은 Z축 회전을 X축보다 먼저 해야한다는 것이다. 왜냐하면 X축으로 먼저 기울인 채 Z축 회전시키면 평면이 찌그러진 채 회전하기 때문이다.

---

3. division 변수를 도입하여 큰 plain을 작은 plain들로 분할(dist와 dx dy로 작은 사각형 점찍고 color 할당)

```
setRectangle 함수

vec4 vertex_pos[4] = { // plain의 네 꼭짓점
    vec4(-0.7 + dist * dx, -0.7 + dist * dy, 0, 1.0),
    vec4(-0.7 + dist * (dx + 1) , -0.7 + dist * dy, 0, 1.0),        // 3   2
    vec4(-0.7 + dist * (dx + 1) , -0.7 + dist * (dy + 1), 0, 1.0),
    vec4(-0.7 + dist * dx, -0.7 + dist * (dy + 1), 0, 1.0),         // 0   1
  };

vec4 vertex_color[2] = {
    vec4(0.7, 0.7, 0.7, 1.0),
    vec4(0.5, 0.5, 0.5, 1.0)
  };

  vertices[index].position = vertex_pos[pos1];    vertices[index].color = vertex_color[color];    index++;
  vertices[index].position = vertex_pos[pos2];    vertices[index].color = vertex_color[color];    index++;
  vertices[index].position = vertex_pos[pos3];    vertices[index].color = vertex_color[color];    index++;
  vertices[index].position = vertex_pos[pos3];    vertices[index].color = vertex_color[color];    index++;
  vertices[index].position = vertex_pos[pos4];    vertices[index].color = vertex_color[color];    index++;
  vertices[index].position = vertex_pos[pos1];    vertices[index].color = vertex_color[color];    index++;

  if (dy == division - 1 && dx == division - 1) { // 모든 점을 다 그렸으면
    dx = 0;
    dy = 0;
    index = 0;
    return; // 초기화 후 탈출
  }

  if (dx == division - 1) { // 행을 다 그렸으면
    dy++; // 열 증가
```

```
    dx = 0; // 행 초기화
    return;
  }
  dx++;
}
```

큰 plain의 한변의 길이를 1.4로 정의했고, 분할된 사각형의 한 변의 길이를 dist로 정의했다.

dx, dy는 각각 분할된 사각형의 행, 열을 뜻한다. (예를들면, dx == 2, dy == 3 이면 왼쪽 맨 아래부터 오른쪽으로 3, 위로 4 올라간 지점에 있는 사각형을 가리킨다.)

작은 사각형마다 왼쪽 아래부터 반시계방향으로 점을 찍도록 vertex_pos의 순서를 정했다.
color는 사각형 하나를 그릴 때마다 밝은회색, 어두운회색 두가지가 번갈아가며 들어가도록 하였다.

사각형을 그려나가다가 한 행의 사각형이 div개가 되면 다음 열로 올라가고, 한 열의 사각형 갯수도 div개가 되면 그림을 그만 그린다.

---

4. ‘1’, ‘2’ key로 division 조절, ‘q’ 누르면 프로그램 종료

```
glutKeyboardFunc(KeyDown); // Key를 입력받으면 실행되는 콜백함수

------------------------------------------------------------

void KeyDown(unsigned char key, int x, int y) {
  switch (key) {

  case '2':
    plain.division++; // 2번을 누르면 division이 증가
    plain.init(); // 점들을 재생성한다
    glutPostRedisplay();
    break;

  case '1':
    if (plain.division <= 2) return; // division이 2 아래로 내려가지 않도록 예외처리
    plain.division--;
    plain.init();
    glutPostRedisplay();
    break;

  case 'q': // q 누르면 프로그램 종료
    exit(0);

  case 'Q': // 대문자로 입력되었을 경우를 방지
    exit(0);
```

KeyDown 함수를 만들고, 특정 Key를 입력받으면 작동하는 콜백함수 glutKeyboardFunc에 넣었다. ‘2’를 누르면 division이 증가하고, ‘1’을 누르면 감소한다.
division이 2 미만으로 내려가지 못하도록 예외처리도 해주었다.

또한 ‘q’를 누르면 프로그램이 종료되도록 하였고, 혹여 대문자로 입력받을 것을 대비해 ‘Q’도 구현해 두었다

▼ 시행착오

2을 입력받아 division을 증가시켰을 때 division은 분명히 증가하지만 화면에 반영되지 않는 문제가 발생했다. 오랜 시간동안 코드를 뜯어본 결과, 여러가지 원인이 있었음을 알게되었다.

- 원인 1

```
plain.init();
```

첫번째 원인은 division이 변화하였지만, 정작 점들을 만들고 보내주는 작업을 새로 하지 않았기 때문이다. 따라서 KeyDown 함수에 키가 호출될 때 마다 plain.init( ) 을 넣어주어서 점을 재생성하고, vao, vbo에 보내주는 작업을 새로 함으로써 해결하였다.

- 원인 2

```
setNumPoints(division); // division을 입력받아 각각
setDist(division);      // NumPoints 와 Dist를 재설정 해주는 함수
```

처음에는 NumPoints(전체 점의 갯수) 와 dist(격자 하나당 넓이) 를 전역변수로 두었었는데, 분명 division은 증가하지만 NumPoints 와 dist는 변하지 않는다는 것을 발견했다. 그래서 이 변수들을 클래스 내의 멤버변수로 넣어주고 setters 함수들을 만들어서 init( )에 넣어줌으로써 division이 변화할 때 NumPoints와 dist에도 반영이 되도록 하여 해결하였다.

- 원인 3

```
if (dy == division - 1 && dx == division - 1) { // 모든 사각형을 그렸으면
  dx = 0;
  dy = 0;
  index = 0;
  return; // 초기화 후 탈출
}
```

위의 두 원인을 해결했을 때 실행하고 division을 증가시키면 자꾸 프로그램이 종료되는 현상이 발생했다. 원인은 동적할당한 vertices 배열의 크기를 넘어가는 할당을 하다가 발생한 오류였다.

처음에는 vertices배열에 점들을 한번만 줄 것이라고 생각했기 때문에setRectangle( ) 함수에서 모든 점들을 다 그렸을 경우를 처리하지 않았는데, division이 생기면서 이제는 새로운 점들을 다시 찍어서 vertices에 넣어줘야 하는 상황이었다.

그래서 모든 점들을 찍었을 때 vertices의 index를 초기화 해주는 코드를 넣어 새롭게 점을 넣어줄 수 있도록 하여 해결하였다.

- 원인 4

```
glutPostRedisplay(); // 그림을 다시 그려준다
```

키가 눌러졌을 때마다 그림을 다시 그려주는 glutPostRedisplay( ) 함수가 없어서 새로운 점이 할당되어도 그림을 그리지 않고 있었다. 따라서 KeyDown 함수에 glutPostRedisplay( )함를 추가하여 해결하였다.

---

5. “SpaceBar” 누르면 회전하도록 구현

```
static int rot = 0; // rotation 모드     0: 정지        1: 회전


case ' ': // spaceBar
    if (rot == 0) { // 정지상태이면
      glutIdleFunc(rotation); // 회전 시작
      glutPostRedisplay();
      rot = 1; // 모드 변경
      break;
    }

    if (rot == 1) { // 회전상태이면
      glutIdleFunc(NULL); // rotation 함수를 빼줌으로써 정지시킴
      glutPostRedisplay();
      rot = 0; // 모드 변경
      break;
    }
```

static변수 rot을 만들어서, 현재 평면이 회전하고 있다면 1, 정지해있다면 0을 저장하도록 하였다. 회전은 아무것도 안해도 계속 호출되는 idle 콜백함수를 이용해야 한다. idle함수에 uniform 타입의 변수 uTime을 증가시키는 rotation 함수를 넣으면 shader에서 uTime을 이용하여 회전이 이루어지므로 평면이 회전하기 시작한다. 스페이스바를 한번 더 눌러서 회전을 정지시킬 때는 idle함수에 들어있던 rotation함수를 제거해줌으로써 uTime이 더이상 증가하지 않도록 해주었다.

---

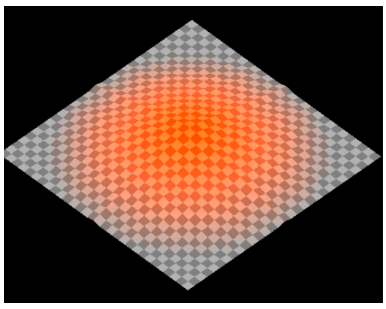
6. vShader에서 wave 생성

```
in vec4 vPosition;
uniform float uTime;
uniform int uDiv;
uniform float uDist;

float angleWave = uTime* 2000;
```

```
float radius = sqrt(vPosition.x * vPosition.x + vPosition.y * vPosition.y);

// -------------------- wave --------------------------

  mat4 wave = mat4(1.0f);

  for(int i = 0; i < uDiv / 2 ; i += 1){

    float dec = 1 - (i / (uDiv / 2.0f)); // i가 증가할수록 1 ~ 0으로 낮아지는 수

    // 가장 안쪽 점 하나
    if(radius <= 0.001f)
      wave[3][2] = 0.2 * sin(radWave);


    if(radius >= uDist * i + 0.001f && radius < uDist * (i + 1) + 0.001f)
      wave[3][2] = 0.2 * dec * sin(2 * radWave - i);

  }
```

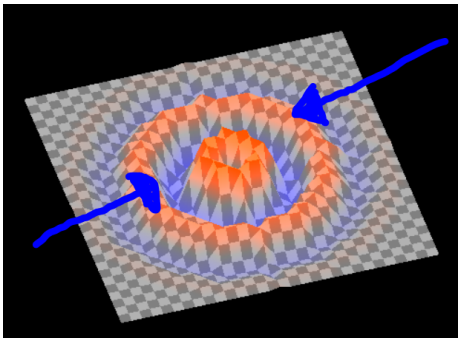
처음에는 각 점의 구조체에 int 타입의 번호를 부여하여서 shader에서 점들을 간편하게 사용하려고 했지만, 사각형마다 중복되는 점이 있어서 번호를 매기기가 힘들었다. 그래서 vShader에서 radius 변수를 정의하고, in으로 받아온 vPosition의 x와 y값을 이용하여 화면의 중심으로부터의 거리를 저장했다.
uniform 타입으로 division와 dist(사각형 하나당 길이) 를 받아와서, 화면의 중심을 시작으로
(div / 2) 까지 dist 간격으로 sin함수에 인자값을 적절하게 넣어주었다. 이 때 인자값을 적절하게 넣는 과정이 굉장히 까다로웠다. sin함수의 인자값에 대해 고려해야 할 점은 다음 두가지였다.

1. 화면의 중심에서부터 자연스럽게 파동이 이어지게 하기 위해서는 각 dist마다 인자값에 차이를 두어야 하는데, 그 차이가 너무 작으면 sin의 값이 중앙이나 화면 가장자리나 차이가 별로 없어지기 때문에, 순차적으로 파동이 일어나지 않고 [그림1]처럼 모든 점이 거의 동시에 오르락 내리락 하게 된다.



[그림1]

2. 화면의 중앙에서 먼저 솟아오르고 그 이후에 가장자리로 파동이 퍼져야 하는데, 인자값을 radwave + i 이런식으로 + i 로 주게 되면 sin그래프상에서 생각해보았을 때 평면의 가장자리의 인자값이 먼저 지나가고, 그 뒤를 평면의 중심의 인자값이 지나가게 된다. 따라서 [그림2]처럼 파동이 바깥에서 안쪽으로 밀려들어오는 모양이 되어버린다. 그러므로 i가 증가할수록 인자값이 감소하도록 하기 위해 radwave - i 를 인자로 주었다.



[그림2]

그리고 평면의 가장자리로 갈수록 파동이 약해지게 구현하기 위해 sin에 1 → 0까지 줄어드는 변수 dec를 만들어 0.2와 곱해줌으로써 중심부에서 가장자리로 갈수록 파동의 진폭이 0.2 → 0으로 줄어들도록 하였다.

또한, 파동의 속도를 조절하기 위해 sin(2 * radWave - i); 처럼 2를 각도에 곱해줌으로써 sin함수의 주기를 조절했다.

▼ 시행착오

wave를 생성할 때 shader 코드를 잘 작성했음에도 불구하고 wave가 작동하지 않는 문제가 발생했다.

```
GLuint uDiv = glGetUniformLocation(prog, "uDiv");
  glUniform1f(uDiv, division);
```

int division을 넘겨주는데 glUniform1f를 사용해서 vShader에서 division을 제대로 읽어오지 못해서 발생한 문제였다. int형을 shader에 넘겨줄 때는 glUniform1i 로 넘겨주어야 한다.

---

7. fShader에서 wave 시 색 변화 구현

```
fcolor = color;

  for(float i = 0; i < 0.2f; i += 0.0001f){

    // z 가 높아질수록 점점 주황색이 된다
    if(position.z > i) fcolor = vec4(color.r + i * 5, color.g - i * 1, color.b - i * 3, 1);

    // z 가 낮아질수록 점점 파란색이 된다
    if(position.z < -i) fcolor = vec4(color.r - i * 1, color.g - i * 1, color.b + i * 3, 1);

  }

}
```

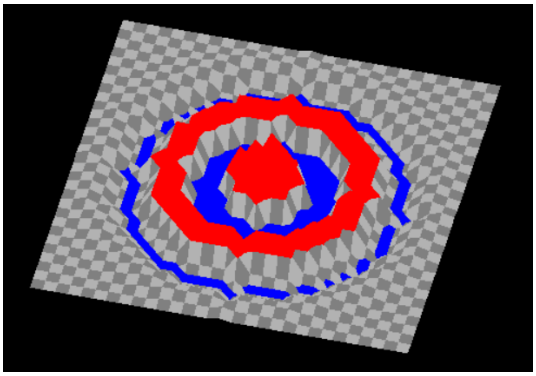
z값이 0보다 크면 주황, 0보다 작으면 파랑으로 점점 바뀌도록 하였다.

▼ 시행착오

- 처음에는 이전에 openGL로 큐브를 만들었을 때 꼭짓점마다 다른 색을 넣었더니 큐브의 면에 자동으로 그라데이션 효과가 나타나는 것을 생각했고, 여기서도 가장 높은 지점과 가장 낮은 지점의 색만 바꿔주면 그 사이에 끼인 면들은 알아서 색이 자연스럽게 변할 것이라고 생각했다.

```
------fShader------

void main(){

  fcolor = color;

  if(position.z >= 0.19 ) fcolor = vec4(1, 0, 0, 1);
  if(position.z <= -0.19) fcolor = vec4(0, 0, 1, 1);
```

하지만 다음과 같이 fShader에서 fcolor에 조건문으로 색을 주게 되면 꼭짓점의 색을 변화시키는 것이 아닌, 완성된 그림을 기준으로 그저 조건에 만족하는 부분의 색 변화시키기 때문에 [그림3] 처럼 될 뿐, 그라데이션 효과가 발생하지 않는다.



[그림3]

결국 for문을 이용하여 z값에 따른 색 변화를 직접 구현하여 해결했다.

- fShader에 position을 넘겨줄 때 z축으로 wave를 적용시킨 이후의 평면을 넘겨주어야 그 평면을 기준으로 z축이 높은부분과 낮은부분의 색을 변화시킬 수 있다.

```
position = wave * vPosition; //  wave를 적용시킨 이후의 평면
```

이를 생각하지 못했을 때는 position = vPosition; 으로 처음 평면을 바로 넘겨주었는데, 이렇게 하면 wave가 생겨도 fShader에 넘어온 평면은 wave가 없는 상태이기 때문에 당연하게도 z값의 변화가 없어서 색 변화가 일어나지 않는다.

---

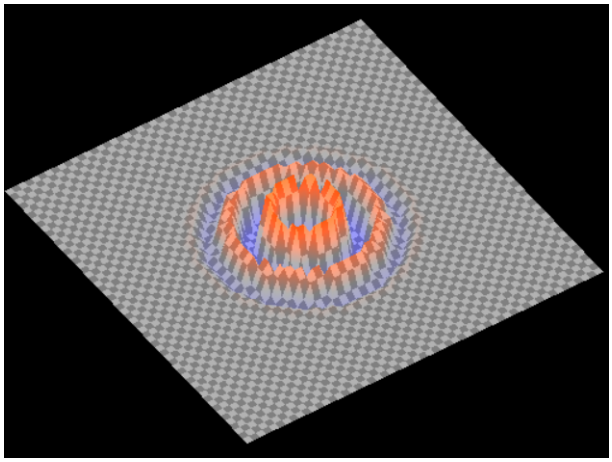
8. ‘w’ 누르면 wave가 작동하도록 구현

```
static int wav = 0; // wave모드      0: not waving     1: waving

------------------------------------------------------------

switch (key) {
  case 'w':

      if (wav == 0) { // not waving
        glUniform1i(uDiv, plain.division);
        wav = 1;
        glutPostRedisplay();
        break;
      }

      if (wav == 1) { // waving
        glUniform1i(uDiv, 0);
        wav = 0;
        glutPostRedisplay();
        break;
      }
}
```

wav 변수를 만들어서 wave가 실행되고 있을때 wav == 1, 아닐때 wav == 0으로 설정하였다. wave를 멈추고 싶을 때 uniform 타입의 uDiv를 0으로 만들어서 vShader에서 for문이 작동하지 않도록 하였다. 원래는 uTime을 조작할 생각이었는데, 그렇게 하면 Z축 중심으로의 회전도 멈추어서 코드를 바꾸었다.

▼ 시행착오

위 코드까지만 구현을 했을 때 발생하는 문제가 있었다. w를 누르면 wave가 생기고 사라지는건 잘 되지만, wave가 작동중일 때 ‘1’ 혹은 ‘2’를 눌러서 div를 변화시키면 [그림4]와 같이 특정 부분에서만 wave가 생기고 그 바깥부분은 wave가 생기지 않는 현상이었다.



[그림 4]

이는 원래 init() 함수 내에 있던 glUniform1i(uDiv, plain.division); 코드를 ‘w’ Key를 입력받는 함수쪽으로 옮겼기 때문에 ‘2’ Key를 눌러서 division이 증가해도 uDiv에는 이전의 division값이 들어가면서 생기는 문제였다.

```
case '2':
    plain.division++; // 2번을 누르면 division이 증가
    plain.init(); // 점들을 재생성한다
    if (wav == 1) // waving
      glUniform1i(uDiv, plain.division);
    glutPostRedisplay();
    break;
```

따라서 위 코드처럼 ‘1’ 혹은 ‘2’를 입력받을 때마다 uDiv를 갱신해줌으로써 해결했다.