7주차 결과보고서

전공: 아트&테크놀로지학과 학년: 4학년 학번: 20191048 이름: 김도솔

**1.**

**ofApp.cpp**

void ofApp::draw() {

(선분 그리기는 생략)

ofSetLineWidth(2); // 선 두께 2로 재설정

if (waterfall\_start\_flag) { // waterfall\_start\_flag가 참인 경우, 즉 ‘s’ 키를 누른 경우

for (unsigned int i = 0; i < wl.size(); i++) { wl 객체의 모든 element에 대해

if (wl[i].path == NULL) continue; // 경로가 null이면 루프의 다음 반복으로 건너뜀

if (wl[i].calc\_complete == 0) // 경로가 계산되지 않은 경우

wl[i].calculate\_path(lineseg, num\_of\_line); // wl[i]의 경로 계산

wl[i].draw(); // wl[i] 객체의 drow 함수 호출하여 흐르는 물 그리기

}

}

}

void ofApp::keyPressed(int key) {

(다른 키 코드 생략)

if (key == 'q') { // ‘q’ 키를 누르면 프로그램 종료

if (!load\_flag) return; // 파일이 로드되지 않은 경우 함수 종료

draw\_flag = 0; // 각종 flag 0으로 설정

waterfall\_start\_flag = 0;

if (lineseg != NULL) { // 라인 세그먼트가 null이 아니면

free(lineseg); // 메모리 할당 해제

lineseg = NULL; // lineseg null로 설정

}

cout << "Memory for line segement has been freed." << endl;

if (dot != NULL) { // dot이 null이 아니면

free(dot); // 메모리 할당 해제

dot = NULL; // dot null로 설정

}

cout << "Memory for dot has been freed." << endl;

\_Exit(0); // 프로그램 종료

}

if (key == 's') { // ‘s’ 키를 누르면 흐르는 물 그리기

if (load\_flag) { // 파일이 로드되었으면

resetWater(); // 물 초기화

for (unsigned int i = 0; i < wl.size(); i++) { // wl 객체의 모든 element에 대해

if (wl[i].calc\_complete == 0) // 경로 계산이 되어있지 않으면

wl[i].calculate\_path(lineseg, num\_of\_line); // 경로 계산

}

if (draw\_flag) { 선분과 점 그리기가 완료 되었으면

waterfall\_start\_flag = 1; // 흐르는 물 그리도록 설정

}

}

}

if (key == 'e') { // ‘e’ 키를 누르면 물 흐르는 것을 멈춤

if (draw\_flag) 선분과 점이 그려졌으면

waterfall\_start\_flag = 0; // 물 그리지 않도록 설정

}

}

void ofApp::resetWater() { // 물 초기화 함수

for (unsigned int i = 0; i < wl.size(); i++) { // wl 객체의 모든 element에 대해

wl[i].start\_dot.x = target\_dot.x; // 시작점을 타겟점으로 설정

wl[i].start\_dot.y = target\_dot.y;

wl[i].reset(); // reset() 멤버 함수 호출

}

}

**water\_line.h**

struct LineSegment { // 선분과 그 선분의 방정식의 계수를 계산하기 위한 변수를 저장하는 구조체

int x1;

int x2;

int y1;

int y2;

double slope;

double x\_coef;

double y\_coef;

double constant;

};

struct Dot { // 점의 위치를 나타내기 위한 구조체

int x1;

int y1;

};

class WaterLine { // 흐르는 물의 경로를 계산하고 그리기 위한 구조체

public:

WaterLine(int num\_of\_line); // 생성자

~WaterLine(); // 소멸자

void reset(); // 초기화 함수

void update();

void draw(); // 경로 그리는 함수

void calculate\_path(LineSegment\* lineseg, int num\_of\_line); // 경로 계산하는 함수

Dot\* path = 0; // 계산된 경로를 저장하는 변수

float uniqueColor\_r, uniqueColor\_g, uniqueColor\_b; // 랜덤 색상 변수

float scale; // 경로 크기 저장 변수

int hexcolor; // 색상 값을 16진수로 저장하는 변수

int path\_idx; // 계산된 경로의 현재 인덱스 저장 변수

int draw\_complete; // 경로 그리기 작업이 완료되었는지 여부를 나타내는 변수

int calc\_complete; // 경로 계산 작업이 완료되었는지 여부를 나타내는 변수

ofPoint start\_dot; // 경로의 시작 위치를 나타내는 변수

float dot\_radius; // 점의 반지름을 나타내는 변수

};

**water\_line.cpp**

void WaterLine::draw() { // 경로 그리는 함수

if (calc\_complete) { // 계산이 완료되면

ofSetLineWidth(5); // 선 두께 5로 설정

ofSetColor(uniqueColor\_r, uniqueColor\_g, uniqueColor\_b); // 색 설정

for (int i = 0; i < path\_idx - 1; i++) { // 생성된 경로를 따라 선 그리기

uniqueColor\_r = ofRandom(0, 100); // 고유한 랜덤 색상값 할당

uniqueColor\_g = ofRandom(0, 100);

uniqueColor\_b = ofRandom(185, 255);

ofSetColor(uniqueColor\_r, uniqueColor\_g, uniqueColor\_b); // 새로 할당된 값으로 색 설정

ofDrawLine(path[i].x1 - 1, path[i].y1 - 1, path[i + 1].x1 + 1, path[i + 1].y1 + 1); // 두 점을 잇는 선 그리기

}

}

draw\_complete = 1; // 그리기 완료

}

void WaterLine::reset() {

// 물줄기마다 랜덤 색상 설정

uniqueColor\_r = ofRandom(0, 100);

uniqueColor\_g = ofRandom(0, 100);

uniqueColor\_b = ofRandom(185, 255);

calc\_complete = 0; // 계산 완료를 나타내는 변수 0으로 설정

draw\_complete = 0; // 그리기 완료를 나타내는 변수 0으로 설정

path\_idx = 0; // 경로 인덱스 0으로 설정

}

void WaterLine::calculate\_path(LineSegment\* lineseg, int num\_of\_line) { // 경로 계산하는 함수

path[path\_idx].x1 = start\_dot.x; // 시작 위치를 경로에 추가

path[path\_idx].y1 = start\_dot.y;

path\_idx++; // 경로 인덱스 +1

for (; start\_dot.y <= ofGetHeight() - 50; start\_dot.y++) { // 이중 반복문을 사용해 모든 라인 세크먼트와 물의 위치를 비교

for (int i = 0; i < num\_of\_line; i++) {

// 물의 위치보다 높은 위치에 있는 선분은 무시

if (start\_dot.y >= lineseg[i].y1 && start\_dot.y >= lineseg[i].y2) continue;

// 현재 위치가 해당 라인 세그먼트 x축 값 사이에 있는지 확인, 만약 해당 선분과 x축 교차점이 없다면 무시하고 다음 라인 세그먼트 검사

if (lineseg[i].x1 < lineseg[i].x2) {

if (start\_dot.x <= lineseg[i].x1 || start\_dot.x >= lineseg[i].x2)

continue;

}

else if (lineseg[i].x1 > lineseg[i].x2) {

if (start\_dot.x <= lineseg[i].x2 || start\_dot.x >= lineseg[i].x1)

continue;

}

// 현재 위치와 해당 라인 세그먼트 사이의 기울기 계산

double temp\_slope = (double)(start\_dot.y - lineseg[i].y1) / (start\_dot.x - lineseg[i].x1);

// 현재 위치가 해당 라인에 속하는 경우 경로에 추가

if (abs(temp\_slope - lineseg[i].slope) <= EPSILON) {

path[path\_idx].x1 = start\_dot.x;

path[path\_idx].y1 = start\_dot.y + 2; // 경로에 현재 점보다 약간 높은 위치의 점을 추가

path\_idx++;

if (lineseg[i].slope < 0) { // 기울기가 음수인 경우 현재 점의 x 좌표를 늘리고

path[path\_idx - 1].x1++;

start\_dot.x = lineseg[i].x1;

start\_dot.y = lineseg[i].y1 - 2;

}

else { // 기울기가 양수인 경우 현재 점의 x 좌표를 줄인다

path[path\_idx - 1].x1--;

start\_dot.x = lineseg[i].x2;

start\_dot.y = lineseg[i].y2 - 2;

}

path[path\_idx].x1 = start\_dot.x;

path[path\_idx].y1 = start\_dot.y;

path\_idx++;

}

}

}

// 마지막 점을 경로에 추가

path[path\_idx].x1 = start\_dot.x;

path[path\_idx].y1 = start\_dot.y;

path\_idx++;

calc\_complete = 1; // 경로 계산 완료

}

WaterLine::WaterLine(int num\_of\_line) { // 생성자

draw\_complete = 0; // 그리기 완료를 나타내는 변수 0으로 설정

calc\_complete = 0; // 계산 완료를 나타내는 변수 0으로 설정

uniqueColor\_r = ofRandom(0, 255); // 랜덤 색상 설정

uniqueColor\_g = ofRandom(0, 255);

uniqueColor\_b = ofRandom(185, 255);

path\_idx = 0; // 경로 인덱스 설정

if (!path) // 경로 변수가 null이 아니면

path = (Dot\*)malloc(sizeof(Dot) \* (2 \* num\_of\_line + 4)); // 메모리 할당

}

WaterLine::~WaterLine() { // 소멸자

free(path); // 메모리 할당 해제

}

**2.**

**Flow Chart**

**도표이(가) 표시된 사진

자동 생성된 설명**