10주차 결과보고서

전공: 수학/컴퓨터공학 학년: 3학년 학번: 20181294 이름: 임승섭

**1.**

우선 version 1.0 폴더와 마찬가지로 “waterline.h”와 “waterline.cpp” 파일을 생성하여 물이 흐르게 하기 위한 변수와 함수들을 선언해주었다. 흐르는 물에 대한 정보를 확인할 수 있다.

1 - 1. “waterline.h”

// 두 선분의 기울기가 거의 유사한 순간을 표현하기 위해 EPSILON이라는 상수를 설정하였다. 값은 0.01

#define EPSILON 0.01

// 선분의 정보를 갖고 있는 LinSegment 구조체를 정의하였다.

struct LineSegment {

int x1;

int y1;

int x2;

int y2;

double slope;

double x\_coef;

double y\_coef;

double constant;

};

// 점의 정보를 갖고 있는 Dot 구조체를 정의하였다.

struct Dot {

int x;

int y;

};

// 물이 흐르는 화면을 구현하기 위해 WaterLine 클래스를 선언하였다.

class WaterLine {

public :

WaterLine(int num\_of\_line); // 생성자. line의 총 개수를 인자로 받는다.

~WaterLine(); // 소멸자. 메모리를 해제시킨다.

void reset(); // 현재 정보를 초기화한다.

void update();

void draw(); // 물이 흐르는 경로를 화면에 그린다.

void calculate\_path(LineSegment\* lineseg, int num\_of\_line); // 물이 흘러가는 경로를 계산한다.

Dot\* path = nullptr; // struct Dot의 path를 선언하였다. 물이 흐르는 지점을 표시하기 위해 사용한다.

float uniqueColor\_r, uniqueColor\_g, uniqueColor\_b; // 물의 색깔을 표시한다.

float scale;

int hexcolor;

int path\_idx;

int path\_idx2;

int draw\_complete; // draw 함수가 완료되면 1로 설정한다. 그림이 그려졌는지 확인한다.

int calc\_complete; //calc 함수가 완료되면 1로 설정한다. 경로가 계산되었는지 확인한다.

ofPoint start\_dot; // path가 시작되는 지점을 나타낸다.

float dot\_radius; // 점의 반지름을 나타낸다.

};

1 - 2. “waterline.cpp”

// waterline의 정보를 초기화하는 함수이다.

void WaterLine::reset() {

// 색을 다시 랜덤으로 초기화하는 과정이다. 물 색깔을 나타내기 위해 b에 큰 가중치를 주었고, 색의 변화가 눈에 보이게 하기 위해 r과 g에도 작은 가중치를 두어 계속 색깔이 변함을 확인할 수 있도록 하였다.

uniqueColor\_r = ofRandom(0, 100);

uniqueColor\_g = ofRandom(0, 100);

uniqueColor\_b = ofRandom(185, 255);

// 초기화하였기 때문에 draw와 calculate는 실행되지 않은 상태이다.

calc\_complete = 0;

draw\_complete = 0;

// 배열 path의 인덱스 번호를 의미하는 path\_idx이다. 초기화하는 과정이기 때문에 0으로 선언한다.

path\_idx = 0;

}

// 생성자이다. reset() 함수와 거의 유사하나, num\_of\_line을 인자로 받아 path에 메모리를 할당할 때 이용한다.

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

// draw와 calculate는 실행되지 않은 상태이다.

draw\_complete = 0;

calc\_complete = 0;

// 위와 마찬가지로 r, g, b를 random으로 설정한다.

uniqueColor\_r = ofRandom(0, 255);

uniqueColor\_g = ofRandom(0, 255);

uniqueColor\_b = ofRandom(285, 255);

// 역시 위와 마찬가지로 path의 인덱스 번호는 0으로 초기화한다.

path\_idx = 0;

// reset()과 다른 부분이다. path를 새로 생성한다. 이 때, malloc보다는 new를 이용하여 메모리 할당을 하도록 한다. 또한, if문을 이용하여 path가 nullptr일 때에만 메모리 할당을 시킨다. 메모리의 크기는 시작점, 끝점, 선분마다 점 두개씩 계산하여 (2 \* num\_of\_line + 4)로 설정해준다.

if (!path)

path = new Dot[2 \* num\_of\_line + 4];

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

}

// 소멸자이다. path의 메모리를 해제한다.

WaterLine::~WaterLine() {

if (!path)

delete[] path;

}

// draw()함수이다. 화면에 물이 흐르는 길을 그린다.

void WaterLine::draw() {

if (calc\_complete) { // calculate함수가 이미 실행되어 경로가 계산되어 있으면 작동한다.

ofSetLineWidth(5); // 선분의 굵기를 5로 설정한다.

ofSetColor(uniqueColor\_r, uniqueColor\_g, uniqueColor\_b); // 선분의 색깔을 설정한다. 위에서 선언한 r, g, b 색깔을 이용한다.

// for문을 이용해서 모든 path의 색깔을 다르게 하여 실제 화면에 그린다. 각 선분의 색깔은 다시 한번 random으로 설정하고, ofDrawLine() 함수를 이용해 시작점과 끝점을 path로 넣어준다. 화면에는 여러 선분이 꺾여서 이어진 상태로 그려지게 된다.

for (int i = 0; i < path\_idx - 1; i++) {

uniqueColor\_r = ofRandom(0, 200);

uniqueColor\_g = ofRandom(0, 200);

uniqueColor\_b = ofRandom(185, 255);

ofSetColor(uniqueColor\_r, uniqueColor\_g, uniqueColor\_b);

ofDrawLine(path[i].x - 1, path[i].y - 1, path[i + 1].x + 1, path[i + 1].y + 1);

}

}

// 그림이 완성되었기 때문에 draw\_complete 변수를 1로 바꿔준다.

draw\_complete = 1;

}

// calculate\_path 함수이다 물이 흘러야 하는 path의 경로를 계산한다.

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

// 처음엔 start\_dot의 위치를 잡아준다. 첫번째 점의 위치를 잡고 path\_idx를 올려준다.

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

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

path\_idx++;

// start\_dot의 y좌표를 올려가면서 loop을 탐색한다.

for (; start\_dot.y <= ofGetHeight() - 50; start\_dot.y++) {

// num\_of\_line 만큼 loop을 돌면서 물의 경로에 있을 선분을 탐색한다.

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

// 물 떨어지는 루트에 선 없는 경우

// 시작점이 line보다 아래에 있는 경우는 고려할 필요가 없다.

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

continue;

// 시작점이 line의 왼쪽 점보다 왼쪽에 있거나 오른쪽 점보다 오른쪽에 있는 경우는 고려할 필요가 없다.

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

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

continue;

}

// 위와 같은 경우이다.

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

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

continue;

}

// 선분이 물이 흐르는 경로에 있을 때 slope 계산

// start\_dot과 line의 점 사이의 기울기를 계산하여 temp\_slope에 저장한다.

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

// 위에서 계산한 기울기와 선분의 기울기가 거의 같아지는 지점에서 path에 시작점을 저장한다. 저장 후 path\_idx는 올려준다.

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

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

path[path\_idx].y = start\_dot.y + 2;

path\_idx++;

cout << "[" << i << "]" << "x: " << start\_dot.x << " " << "y: " << start\_dot.y << endl;

// line의 기울기가 음수인 경우에는 방금 설정한 path의 값을 1 올려준다. 이후 start\_dot을 line의 (x1, y1 - 2)로 선언한다.

if (lineseg[i].slope < 0) {

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

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

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

}

// line의 기울기가 0 이상인 경우에는 방금 설정한 path의 값을 1 낮춰준다. 이후 start\_dot을 line의 (x2, y2 - 2)로 선언한다.

else {

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

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

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

}

// 새로운 path 값을 선언하고 path\_idx를 1 올린다.

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

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

path\_idx++;

cout << "[" << i << "]" << "x: " << start\_dot.x << " " << "y: " << start\_dot.y << endl;

}

}

}

// loop이 모두 끝난 후 path의 값을 선언하고 path\_idx를 1 올린다.

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

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

path\_idx++;

// 경로 계산이 완료되었기 때문에 calc\_complete 값을 1로 바꿔준다.

calc\_complete = 1;

}

다음은 “ofApp.h”와 “ofApp.cpp” 파일을 만들어서 waterfall 전반에 대한 정보를 설정해주었다. 물이 떨어지기 전의 화면 구성과 키보드 입력을 받아들이는 등의 기능을 수행한다.

1 - 3. “ofApp.h”

기본적인 함수와 변수들을 선언해주었다.

class ofApp : public ofBaseApp{

public:

void setup();

void update();

void draw();

void resetWater(); // waterline의 시작점을 설정해주는 역할을 한다.

void keyPressed(int key);

void keyReleased(int key);

void mouseMoved(int x, int y );

void mouseDragged(int x, int y, int button);

void mousePressed(int x, int y, int button);

void mouseReleased(int x, int y, int button);

void mouseEntered(int x, int y);

void mouseExited(int x, int y);

void windowResized(int w, int h);

void dragEvent(ofDragInfo dragInfo);

void gotMessage(ofMessage msg);

/\* WaterFall-related member variables Regions \*/

LineSegment\* lineseg = nullptr; // LineSegment 타입의 lineseg를 선언한다. 각 선분을 나타낸다.

Dot\* dot = nullptr; // Dot 타입의 dot을 선언한다. 물이 나와야 하는 각 점을 나타낸다.

// flag variables

int draw\_flag; // 화면에 선분과 점이 그려졌는지를 확인하는 변수이다.

int load\_flag; // input 파일이 로드되었는지 확인하는 변수이다.

// Line segment and dot related variables

int num\_of\_line, num\_of\_dot; // 화면에 표시될 선분과 점의 개수를 나타내는 변수이다.

float dot\_diameter; // 점의 지름을 나타내는 변수이다.

int waterfall\_start\_flag; // 물이 흐르는 화면이 시작되었는지 확인하는 변수이다.

int selection\_dot; // 원하는 점(흐르는 물이 시작)을 나타내는 변수이다. index값으로 이용된다.

ofPoint target\_dot; // 현재 target으로 잡고 있는(빨간색으로 나타남) 점을 나타내는 변수이다.

float water\_radius;

vector <WaterLine> wl; // WaterLine class 의 vector wl을 선언한다.

/\* WaterFall-related member functions \*/

void processOpenFileSelection(ofFileDialogResult openFileResult);

};

1 - 4. “ofApp.cpp”

void ofApp::setup(){

ofSetFrameRate(30); // 초당 30프레임으로 맞춰준다.

// We still want to draw on a black background, so we need to draw

// the background before we do anything with the brush

ofBackground(255,255,255); // 배경색은 검정색으로 설정한다

ofSetLineWidth(4); // 선의 두께는 4로 설정한다.

// 초기설정이기 때문에 flag변수들과 점 인덱스 값은 0을 주고, 점의 지름은 20을 준다.

draw\_flag = 0;

load\_flag = 0;

dot\_diameter = 20.0f;

waterfall\_start\_flag = 0;

selection\_dot = 0;

}

void ofApp::draw(){

ofSetColor(127,23,31); // 갈색으로 설정한다.

// Draw shapes for ceiling and floor

ofDrawRectangle(0, 0, 1024, 40); // Top left corner at (50, 50), 100 wide x 100 high

ofDrawRectangle(0, 728, 1024, 40); // Top left corner at (50, 50), 100 wide x 100 high

ofSetLineWidth(5);

// 먼저 선분과 점을 화면에 그린다.

if (draw\_flag) { // draw\_flag가 1이 되었을 때 화면에 그림을 시작한다.

int i

// 선분의 개수만큼 loop을 돌면서 각 선분을 화면에 그린다.

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

// ofDrawLine을 이용해서 선분의 시작점과 끝점을 이용한다.

ofDrawLine(lineseg[i].x1, lineseg[i].y1, lineseg[i].x2, lineseg[i].y2);

}

// 점의 개수만큼 loop을 돌면서 각 점을 화면에 그린다.

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

if (selection\_dot == i) // 현재 선택한 점의 인덱스와 같을 땐 빨간색으로 표시한다.

ofSetColor(255, 0, 0);

else // 그렇지 않으면 검정색으로 표시한다.

ofSetColor(0);

// ofDrawCircle을 이용하여 점의 좌표와 지름의 길이를 이용한다.

ofDrawCircle(dot[i].x, dot[i].y, dot\_diameter / 2);

}

}

ofSetLineWidth(2);

/\*---------------------week2----------------------\*/

// 두번째 주차에서는 물이 떨어지는 모습을 화면에 그렸다

if (waterfall\_start\_flag) { // waterfall\_flag가 1이 되었을 때 화면에 그림을 시작한다.

// 벡터로 선언했던 wl의 size만큼 loop을 돌면서 물줄기를 그린다.

for (int i = 0; i < wl.size(); i++) {

if (wl[i].path == nullptr) continue; // path가 nullptr일 땐 그릴 게 없다.

// 이 부분에서 시간을 많이 낭비했다. cal\_complete가 0이면 calculate\_path 함수를 사용해서 물줄기의 경로를 계산해주면 되는데, if 조건문 안에 draw 함수까지 넣어두니까, 물줄기가 한 번 보이고 사라지는 현상이 계속되었다. 그래서 if 조건문에는 calculate\_path 함수 하나만 넣어주고, draw 함수는 조건문 밖에 위치시켜서 문제를 해결하였다.

if (wl[i].calc\_complete == 0) //

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

wl[i].draw();

cout << "draw good" << endl;

}

}

}

void ofApp::keyPressed(int key){

if (key == 'v') { // v가 눌렸을 때는 화면을 캡쳐한다.

// HACK: only needed on windows, when using ofSetAutoBackground(false)

glReadBuffer(GL\_FRONT);

ofSaveScreen("savedScreenshot\_"+ofGetTimestampString()+".png");

}

if (key == 'q'){ // q가 눌렸을 때는 프로그램을 종료한다.

if (!load\_flag) return; // load가 되지 않았을 때는 그대로 종료하면 된다.

// Reset flags

// flag들을 0으로 리셋한다.

draw\_flag = 0;

waterfall\_start\_flag = 0;

// Free the dynamically allocated memory exits.

// 동적 할당했던 메모리들을 해제한다. lineseg와 dot의 메모리를 해제하고, nullptr로 초기화한다.

if (!lineseg) {

delete[] lineseg;

lineseg = nullptr;

}

if (!dot) {

delete[] dot;

dot = nullptr;

}

cout << "Dynamically allocated memory has been freed." << endl;

\_Exit(0); // 완전히 종료한다.

}

if (key == 'd'){ // d가 눌렸을 때는 화면에 점과 선분을 그려준다.

if( !load\_flag) return; // load가 되지 않았을 때는 그대로 종료한다.

draw\_flag = 1; // draw\_flag를 1로 만들어서 draw() 함수가 실행되도록 한다.

// 현재 선택한 점을 target\_dot으로 설정한다.

target\_dot.x = dot[selection\_dot].x;

target\_dot.y = dot[selection\_dot].y;

}

/\*------------------------------week2----------------------------\*/

if (key == 's'){ // s가 눌렸을 때는 선택한 점에서 물이 나오는 화면을 그린다.

if (load\_flag) { // load가 되었을 때만 그림을 그린다.

resetWater(); // wl의 시작점을 현재 target\_dot으로 설정하는 부분이다.

if (draw\_flag) { // draw\_flag가 1일 때, 즉 이미 화면에 점과 선분이 그려진 상태에서 물줄기를 그린다.

waterfall\_start\_flag = 1; // 물줄기를 그리기 위해 waterfall\_start\_flag를 1로 바꿔준다.

cout << "water line start" << endl;

}

}

}

if (key == 'e'){ // e가 눌렸을 때는 흐르고 있는 물줄기를 멈춘다.

if (draw\_flag) { // 화면에 점과 선분이 그려져 있는 상태에서 작동한다

waterfall\_start\_flag = 0; // 물줄기를 멈추기 위해 waterfall\_start\_flag를 0으로 바꿔준다.

cout << "water line stop" << endl;

}

}

}

void ofApp::keyReleased(int key){

if( key == 'l'){ // l이 눌렸을 때는 input 파일을 업로드한다.

// Open the Open File Dialog

ofFileDialogResult openFileResult= ofSystemLoadDialog("Select a only txt for Waterfall");

// Check whether the user opened a file

if( openFileResult.bSuccess){

ofLogVerbose("User selected a file");

// We have a file, so let's check it and process it

processOpenFileSelection(openFileResult);

load\_flag = 1; // 파일 로드가 완료되면 load\_flag를 1로 바꿔준다.

selection\_dot = 0; // 선택된 점은 우선 초기값인 0으로 잡아준다.

}

}

/\* COMSIL1-TODO 4: This is selection dot control part.

You can select dot in which water starts to flow by left, right direction key (<- , ->).

\*/

if (key == OF\_KEY\_RIGHT){ // 오른쪽 방향키가 눌렸을 때는 현재 선택한 점을 오른쪽으로 옯긴다.

waterfall\_start\_flag = 0; // 혹시 물이 흐르고 있었다면 멈추기 위해 waterfall\_start\_flag를 0으로 설정한다.

selection\_dot++; // selection\_dot, dot의 인덱스를 한 칸 증가시켜서 오른쪽의 점이 선택되도록 한다.

if (selection\_dot == num\_of\_dot) // 만약 오른쪽 끝까지 갔다면 다시 맨 왼쪽으로 초기화시켜준다.

selection\_dot = 0;

// target\_dot을 새로 바뀐 점으로 재설정한다.

target\_dot.x = dot[selection\_dot].x;

target\_dot.y = dot[selection\_dot].y;

cout << "Selcted Dot Coordinate is (" << dot[selection\_dot].x << ", " << dot[selection\_dot].y <<")" << endl;

resetWater();

}

if (key == OF\_KEY\_LEFT){ // 왼쪽 방향키가 눌렸을 때는 현재 선택한 점을 왼쪽으로 옮긴다.

waterfall\_start\_flag = 0; // 혹시 물이 흐르고 있었다면 멈추기 위해 waterfall\_start\_flag를 0으로 설정한다.

selection\_dot--; // selection\_dot, dot의 인덱스를 한 칸 감소시켜서 왼쪽의 점이 선택되도록 한다.

if (selection\_dot == -1) // 만약 맨 왼쪽까지 갔다면 다시 맨 오른쪽으로 초기화시켜준다.

selection\_dot = num\_of\_dot - 1;

// target\_dot을 새로 바뀐 점으로 재설정한다.

target\_dot.x = dot[selection\_dot].x;

target\_dot.y = dot[selection\_dot].y;

cout << "Selcted Dot Coordinate is (" << dot[selection\_dot].x << ", " << dot[selection\_dot].y << ")" << endl;

resetWater();

}

}

void ofApp::resetWater() { // wl의 현재 start\_dot을 target\_dot으로 설정한다. 물줄기의 경로를 그리는 데 필요한 함수이다.

for (unsigned int i = 0; i < wl.size(); i++) {

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

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

wl[i].reset();

}

}

void ofApp::processOpenFileSelection(ofFileDialogResult openFileResult) {

//Path to the comma delimited file

//string fileName = "input.txt";

// ‘input.txt’ 파일을 오픈한다. 파일이 존재하지 않을 때는 에러 메세지를 출력하고, 정상적으로 찾았을 때는 찾았다는 메세지를 출력한다.

string fileName = openFileResult.getName();

ofFile file(fileName);

if( !file.exists()) cout << "Target file does not exists." << endl;

else cout << "We found the target file." << endl;

ofBuffer buffer(file);

/\* This variable is for indicating which type of input is being received.

IF input\_type == 0, then work of getting line input is in progress.

IF input\_type == 1, then work of getting dot input is in progress.

\*/

int input\_type = 0;

/\* COMSIL1-TODO 1 : Below code is for getting the number of line and dot, getting coordinates.

You must maintain those information. But, currently below code is not complete.

Also, note that all of coordinate should not be out of screen size.

However, all of coordinate do not always turn out to be the case.

So, You have to develop some error handling code that can detect whether coordinate is out of screen size.

\*/

bool index = false;

int idx = 0;

// Read file line by line

for (ofBuffer::Line it = buffer.getLines().begin(), end = buffer.getLines().end(); it != end; ++it) {

string line = \*it;

// Split line into strings

vector<string> words = ofSplitString(line, " "); // line을 공백으로 분리한 것을 words에 저장한다.

if( words.size() == 1){ // size가 1이라면 선분의 개수 또는 점의 개수를 나타낸다.

if( input\_type == 0){ // 선분의 개수를 나타낸다.

num\_of\_line = atoi(words[0].c\_str()); // 정수형으로 변환시켜 num\_of\_line에 저장한다.

cout << "The number of line is: " << num\_of\_line << endl;

// num\_of\_line 만큼 lineseg에 동적 할당을 해준다.

if (!lineseg)

lineseg = new LineSegment[num\_of\_line];

}

else{ // 점의 개수를 나타낸다.

num\_of\_dot = atoi(words[0].c\_str()); // 정수형으로 변환시켜 num\_of\_dot에 저장한다.

cout << "The number of dot is: " << num\_of\_dot << endl;

// num\_of\_dot만큼 dot에 동적 할당을 해준다.

if (!dot)

dot = new Dot[num\_of\_dot];

}

// 선분의 개수가 없다면 input\_type을 1로 바꿔준다.

if (num\_of\_line == 0)

input\_type = 1;

}

else if (words.size() >= 2){ // size가 2 이상이라면 선분 또는 점의 좌표를 나타낸다.

int x1,y1,x2,y2; // 좌표를 나타내는 변수로 사용한다.

if( input\_type == 0){ // 선분의 정보를 나타낸다.

// 각 좌표를 정수로 변환시켜서 lineseg[idx]의 좌표로 저장한다.

x1 = atoi(words[0].c\_str());

y1 = atoi(words[1].c\_str());

x2 = atoi(words[2].c\_str());

y2 = atoi(words[3].c\_str());

lineseg[idx].x1 = x1;

lineseg[idx].y1 = y1;

lineseg[idx].x2 = x2;

lineseg[idx].y2 = y2;

// 선분을 그릴 수 없는 위치의 좌표라면 에러 메세지를 출력하고 함수를 종료한다.

if (lineseg[idx].x1 < 0 || lineseg[idx].x1 > ofGetWidth()) {

cout << "out of range : " << lineseg[idx].x1 << endl;

return;

}

else if (lineseg[idx].y1 < 0 || lineseg[idx].y1 > ofGetHeight()) {

cout << "out of range : " << lineseg[idx].y1 << endl;

return;

}

else if (lineseg[idx].x2 < 0 || lineseg[idx].x2 > ofGetWidth()) {

cout << "out of range : " << lineseg[idx].x2 << endl;

return;

}

else if (lineseg[idx].y2 < 0 || lineseg[idx].y2 > ofGetWidth()) {

cout << "out of range : " << lineseg[idx].y2 << endl;

return;

}

double tmp\_constant = 0.0f;

// 선분의 좌표를 이용해서 기울기를 계산하고 저장한다.

lineseg[idx].slope = (double)(lineseg[idx].y2 - lineseg[idx].y1) / (int)(lineseg[idx].x2 - lineseg[idx].x1);

// y2 - y1 값을 x\_coef에 저장한다.

lineseg[idx].x\_coef = (double)(lineseg[idx].y2 - lineseg[idx].y1);

// x2 - x1 값의 부호를 바꿔서 y\_coef에 저장한다.

lineseg[idx].y\_coef = (double)(lineseg[idx].x2 - lineseg[idx].x1) \* (-1);

tmp\_constant = lineseg[idx].slope \* (-1) \* lineseg[idx].x1 + lineseg[idx].y1;

lineseg[idx].constant = (double)tmp\_constant \* (lineseg[idx].x2 - lineseg[idx].x1);

// index를 증가시키다 num\_of\_line을 넘어가면 다시 0으로 초기화시킨다.

idx++;

if (idx >= num\_of\_line) {

input\_type = 1;

idx = 0;

}

}

else{ // 점의 좌표를 나타낸다.

index = true;

// 정수형으로 바꾼 값을 dot[idx]의 x1, y1에 저장한다.

x1 = atoi(words[0].c\_str());

y1 = atoi(words[1].c\_str());

dot[idx].x = x1;

dot[idx].y = y1;

idx++;

if (idx >= num\_of\_dot)

input\_type = 0;

}

} // End of else if.

} // End of for-loop (Read file line by line).

// vector wl을 생성한다. num과 num\_of\_line 변수를 이용하고, resetWater함수를 이용해 초기화시켜준다.

int num = 50;

wl.assign(num, WaterLine(num\_of\_line));

dot\_diameter = 20.0f;

for (unsigned int i = 0; i < wl.size(); i++) {

wl[i].start\_dot.x = dot[selection\_dot].x;

wl[i].start\_dot.y = dot[selection\_dot].y;

wl[i].dot\_radius = dot\_diameter / 2;

}

resetWater();

}

**2.**

draw() 함수의 flow chart

(1).

초기값 설정(ofSetColor, ofDrawRectangle, ofSetLineWidth)

↓

(2).

draw\_flag가 1인지 확인 -> (no : (4)로 이동)

↓ (yes)

(3).

num\_of\_line만큼 선분 그림 (ofDrawLine)

num\_of\_dot만큼 점 그림(ofDrawCircle)

↓

(4).

waterfall\_start\_flag가 1인지 확인 -> (no :

↓ (yes)

(5).

wl calculate(calc\_complete가 0이면) 함수 호출 후

wl draw() 통해 물줄기 화면에 그림.