**7주차 결과보고서**

**전공: 경영학과 학년: 3학년 학번: 20190963 이름: 한다현**

**1.**

**ofSetFrameRate(15); // Limit the speed of our program to 60 frames per second**

**// 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);**

**ofSetLineWidth(4);**

**currentIndex = 0;**

**draw\_flag = 0;**

**load\_flag = 0;**

**waterfall\_start\_flag = 0;**

**selection\_dot = 0;**

**가장 먼저 setup() 함수에서 배경색을 흰색, 선의 굵기를 4로 설정하고, currentIndex, draw\_flag, load\_flag, waterfall\_start\_flag, selection\_dot을 모두 0으로 초기화 한다.**

**이후 L키를 눌렀을 때 input 파일을 위한 다이얼로그 창을 생성하는 것을 구현한다.**

**if( key == 'l'){**

**// 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, check it and process it**

**processOpenFileSelection(openFileResult);**

**load\_flag = 1;**

**}**

**}**

**keyReleased 함수를 통해 L키를 누르면 다이얼로그 창이 열린다. 이 때 성공적으로 열린다면 processOpenFileSelection이라는 함수를 실행하고 load\_flag의 값을 1로 바꾼다.**

**if( !file.exists()){**

**cout << "Target file does not exists." << endl;**

**}**

**else{**

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

**}**

**processOpenFileSelection 함수에서는 먼저 위 코드를 통해 파일의 존재 여부를 검사한다.**

**int input\_flag=0;**

**// Idx is a variable for index of array.**

**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, " ");**

**if(words.size() == 1){**

**if( !input\_flag){ // Input for number of line**

**num\_of\_line = atoi(words[0].c\_str());**

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

**if( !lineseg)**

**lineseg = (LineSegment\*)malloc(sizeof(LineSegment) \* num\_of\_line);**

**}**

**else{ // Input for number of dot**

**num\_of\_dot = atoi(words[0].c\_str());**

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

**if( !dot)**

**dot = (Dot\*)malloc(sizeof(Dot) \* num\_of\_dot);**

**}**

**}**

**input\_flag라는 변수를 생성해서 input\_flag가 0이면 line에 대한 작업을 수행하고, 1이면 dot에 대한 작업을 실행한다. 이후 파일을 1줄씩 읽어들여서 각 줄에 존재하는 단어들을 words라는 변수에 저장한다. words.size()의 값이 1이라는 것은 line이나 dot의 개수를 알려주는 것이다. words.size()가 1일 때 input\_flag가 0이면 line의 개수를 num\_of\_line에 저장하고 이를 출력한다. 또한 lineseg라는 구조체 배열의 메모리를 동적으로 할당 받는다. input\_flag가 1이면 dot의 개수를 num\_of\_dot에 저장하고 이를 출력하며 dot이라는 구조체 배열의 메모리를 동적으로 할당 받는다. 이 때 LineSegment와 Dot이라는 구조체가 사용되었다. 이 두 개의 구조체는 water\_line.h 파일에 선언하는데 각각의 구조체는 다음과 같다.**

**typedef struct LineSegment {**

**int x1;**

**int y1;**

**int x2;**

**int y2;**

**double slope;**

**double x\_coef;**

**double y\_coef;**

**double constant;**

**};**

**typedef struct Dot {**

**int x1;**

**int y1;**

**};**

**LineSegmanet 구조체에서는 첫 번째 x, y 좌표를 저장하는 x1, y1과 두 번째 x, y 좌표를 저장하는 x2, y2가 있고, 선의 기울기를 저장하는 slope, x의 계수, y의 계수을 저장하는 x\_coef, y\_coef, 그리고 상수를 저장하는 constant가 있다. Dot 구조체에는 점의 좌표를 저장하는 x1, y1이 있다.**

**word.size()가 2 이상이면 다음의 코드를 실행한다.**

**else if (words.size() >= 2){**

**// Input for actual lines.**

**if( !input\_flag){**

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

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

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

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

**// Out-of-range error handling**

**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 > ofGetHeight()){**

**cout << "Out-of-range: " << lineseg[idx].y2 << endl;**

**return;**

**}**

**input\_flag가 0이면 lineseg에 라인의 두 끝 점의 좌표를 x1, y1, x2, y2에 저장한다. 그 후 선의 좌표가 화면의 범위를 벗어나면 에러 메시지를 출력한다.**

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

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

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

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

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

**idx++;**

**// Set the flag**

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

**input\_flag = 1;**

**idx = 0;**

**}**

**}**

**이 후 slope, x\_coef, y\_coef, constant를 계산한 후 배열의 인덱스를 1 증가시킨다. 이 때 인덱스가 num\_of\_line의 값보다 크면 input\_flag를 1로 변경한 후 인덱스를 초기화한다.**

**else{ // Input for actual dots.**

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

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

**// Debug output**

**//cout << dot[idx].x1 << " " << dot[idx].y1 << " " << endl;**

**idx++;**

**// Unset the flag**

**if( idx >= num\_of\_dot)**

**input\_flag = 0;**

**}**

**}**

**input\_flag가 0이 아니면 점의 좌표를 x1, y1에 저장한 후 인덱스를 1 증가시킨다. 이 때 인덱스가 num\_of\_dot의 값보다 크면 input\_flag를 0으로 초기화한다.**

**다음은 void ofApp::keyPressed(int key)함수의 일부이다.**

**if (key == 'd'){**

**if( !load\_flag ) return;**

**draw\_flag = 1;**

**target\_dot.x = dot[selection\_dot].x1;**

**target\_dot.y = dot[selection\_dot].y1;**

**}**

**d 키가 눌리면 먼저 load\_flag가 0인지 확인하고 0이라면 함수를 종료한다. 0이 아니면 draw\_flag를 1로 설정하고, target\_dot의 x와 y에 각각 dot의 selection\_dot 인덱스에 있는 x1, y1를 저장한다.**

**다음은 draw() 함수 내부의 코드이다.**

**ofSetColor(127,23,31); // Set the drawing color to brown**

**// 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);**

**그림의 색을 brown으로 설정한 후 두 개의 직사각형을 그리고 선의 굵기를 5로 설정한다.**

**if( draw\_flag){**

**int i;**

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

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

**}**

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

**if( selection\_dot == i)**

**ofSetColor(255, 0, 0);**

**else**

**ofSetColor(0);**

**ofDrawCircle(dot[i].x1,dot[i].y1, dot\_diameter/2);**

**}**

**}**

**draw\_flag가 1이면 그림을 그린다. num\_of\_line의 값만큼 반복하며 모든 라인을 그린다. 또한 num\_of\_dot의 값만큼 반복하며 점을 그리는데, selection\_dot에 해당하는 점은 빨간색으로, 그 외의 점들은 검정색으로 그린다.**

**void ofApp::keyReleased(int key) 함수로 돌아가서, 사용자가 오른쪽 화살표를 입력하면 다음의 코드를 실행한다.**

**if( key == OF\_KEY\_RIGHT){**

**waterfall\_start\_flag = 0;**

**selection\_dot++;**

**if(selection\_dot == num\_of\_dot)**

**selection\_dot = 0;**

**target\_dot.x = dot[selection\_dot].x1;**

**target\_dot.y = dot[selection\_dot].y1;**

**cout << "Selcted Dot Coordinate is (" << target\_dot.x << ", " << target\_dot.y << ")" << endl;**

**resetWater();**

**}**

**waterfall\_start\_flag를 0으로 설정하고, selection\_dot의 값을 1 증가시킨다. 이 때 selection\_dot의 값이 num\_of\_dot의 값과 같으면 selection\_dot을 다시 0으로 설정하고, target\_dot의 x와 y의 값을 변경된 selection\_dot의 좌표로 다시 저장한 후 이 좌표를 출력하며 resetWater() 함수를 호출한다. resetWater() 함수는 나중에 설명한다.**

**사용자가 왼쪽 화살표를 입력하면 다음 코드를 실행한다.**

**if( key == OF\_KEY\_LEFT){**

**waterfall\_start\_flag = 0;**

**selection\_dot--;**

**if (selection\_dot < 0)**

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

**target\_dot.x = dot[selection\_dot].x1;**

**target\_dot.y = dot[selection\_dot].y1;**

**cout << "Selcted Dot Coordinate is (" << target\_dot.x << ", " << target\_dot.y << ")" << endl;**

**resetWater();**

**}**

**waterfall\_start\_flag를 0으로 설정하고, selection\_dot의 값을 1 감소시킨다. 이 때 selection\_dot의 0보다 작으면 selection\_dot이 마지막 점을 가리키도록 하고, target\_dot의 x와 y의 값을 변경된 selection\_dot의 좌표로 다시 저장한 후 이 좌표를 출력하며 resetWater() 함수를 호출한다.**

**사용자가 q를 입력하면 void ofApp::keyPressed(int key)에서 다음의 코드를 실행한다.**

**if( key == 'q'){**

**if( !load\_flag ) return;**

**// Reset flags**

**draw\_flag = 0;**

**waterfall\_start\_flag = 0;**

**// Free the Dynamic\_allocation entities.**

**if( lineseg != NULL){**

**free(lineseg);**

**lineseg = NULL;**

**}**

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

**if( dot != NULL){**

**free(dot);**

**dot = NULL;**

**}**

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

**\_Exit(0);**

**}**

**load\_flag가 0이 아니며 draw\_flag와 waterfall\_start\_flag를 0으로 저장하고 lineseg와 dot에 할당된 메모리를 해제하며 메모리가 해제되었다는 문구를 출력한다.**

**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 = ofRandom(dot[selection\_dot].x1-dot\_diameter/2, dot[selection\_dot].x1+dot\_diameter/2);**

**wl[i].start\_dot.x = dot[selection\_dot].x1;**

**wl[i].start\_dot.y = dot[selection\_dot].y1;**

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

**}**

**resetWater();**

**WaterLine 클래스의 객체를 생성한다. WaterLine의 생성자는 다음과 같다.**

**draw\_complete = 0;**

**calc\_complete = 0;**

**uniqueColor\_r = ofRandom(0,255);**

**uniqueColor\_g = ofRandom(0,255);**

**uniqueColor\_b = ofRandom(185, 255);**

**path\_idx = 0;**

**if(path)**

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

**draw\_complete, calc\_complete를 0으로 초기화하고, uniqueColor\_r,g,b를 초기화한다. path\_idx를 0으로 설정한 후 path가 null이 아니면 path를 동적으로 할당 받는다. 이렇게 생성된 WaterLine 클래스의 객체와 num을 WL에 할당한다. 이후 dot\_diameter를 20.0f로 설정하고 WL의 start\_dot의 x와 y에 dot[selection\_dot].x1, y1을 저장하고 WL의 dot.radius를 dot\_diameter를 2로 나눠가며 저장한다. 그 후 resetWater() 함수를 호출한다.**

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

**//wl[i].start\_dot.x = ofRandom(dot[selection\_dot].x1-dot\_diameter/2, dot[selection\_dot].x1+dot\_diameter/2);**

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

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

**wl[i].reset();**

**}**

**resetWater() 함수에는 WL.size만큼 반복하며 wL의 start\_dot.x, start\_dot.y를 target\_dot x, y로 설정하고 reset() 함수를 호출한다. reset() 함수는 다음과 같다.**

**uniqueColor\_r = ofRandom(0,100);**

**uniqueColor\_g = ofRandom(0,100);**

**uniqueColor\_b = ofRandom(185, 255);**

**calc\_complete = 0;**

**draw\_complete = 0;**

**path\_idx = 0;**

**색깔을 초기화하고, calc\_complete, draw\_complete, path\_idx를 모두 0으로 초기화한다.**

**다음은 S 키를 눌렀을 때 물이 흐르는 경로를 표시하는 코드를 작성한다.**

**if (key == 's'){**

**if( load\_flag ){**

**resetWater();**

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

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

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

**}**

**if(draw\_flag){**

**waterfall\_start\_flag = 1;**

**}**

**}**

**}**

**먼저 S키가 눌렸을 때 load\_flag가 0이 아니면 resetWater() 함수를 실행하고, calc\_complete가 0이면 경로를 계산하는 calculate\_path함수를 실행한다. 이후 draw\_flag가 0이 아니면 waterfall\_start\_flag에 1을 저장한다.**

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

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

**path\_idx++;**

**calculate\_path 함수에서는 먼저 path\_idx에 해당하는 x1, y1에 start\_dot.x와 y를 저장한고 path\_idx를 1 증가시킨다.**

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

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

**// 1) Ignore line that located higher than water particle.**

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

**// 2) Check whether the dot point exists between the each end of line segment.**

**// If pos.x exists btw. y1 and y2 then, eventually hit the line segment.q**

**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;**

**}**

**// Slope 계산**

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

**이후 물이 흐르는 경로와 만나는 선을 찾는데, 물이 흐르는 경로보다 위에 위치한 선은 무시하고, 점이 선의 양 끝점 사이에 존재하는지 확인한다. 존재한다면 기울기를 계산하여 temp\_slope에 저장한다.**

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

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

**path[path\_idx].y1 = start\_dot.y+2;**

**path\_idx++;**

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

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

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

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

**}**

**else{**

**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++;**

**}**

**temp\_slope와 선의 기울기의 차이가 0.01보다 작거나 같으면 path\_idx에 해당하는 x1, y1의 값을 변경한 후 path\_idx를 1 증가시킨다. 선의 기울기가 0보다 작으면 물이 오른쪽으로 흐르는 것을 표현할 수 있도록 path[path\_idx-1].x1의 값을 1 증가시키고 start\_dot.x와 start\_dot.y의 값을 변경한다. 반대로 선의 기울기가 0보다 크거나 같으면 물이 왼쪽으로 흐르는 것을 표현할 수 있도록 관련 변수들의 값을 변경한다. 그 후 path\_idx에 해당하는 x1, y1을 start\_dot.x, y로 저장하고 path\_idx를 1 증가시킨다.**

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

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

**path\_idx++;**

**calc\_complete = 1;**

**경로의 계산이 끝났다면 path\_idx에 해당하는 x1, y1을 start\_dot.x, y로 저장하고 path\_idx를 1 증가시키며 calc\_complete에 1을 저장한다.**

**다음은 ofApp.cpp의 draw 함수 중 일부이다.**

**ofSetLineWidth(2);**

**if(waterfall\_start\_flag){**

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

**if( wl[i].path == NULL) continue;**

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

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

**wl[i].draw();**

**}**

**}**

**라인의 굵기를 2로 설정하고 waterfall\_start\_flag가 0이 아니면 반복문을 실행한다. path가 null이면 continue, calc\_complete가 0이면 calculate\_path를 실행하고 WaterLine 클래스의 draw() 함수를 실행한다. draw() 함수는 다음과 같다.**

**if( calc\_complete){**

**ofSetLineWidth(5);**

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

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

**uniqueColor\_r = ofRandom(0,100);**

**uniqueColor\_g = ofRandom(0,100);**

**uniqueColor\_b = ofRandom(185, 255);**

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

**ofDrawLine(path[i].x1-1, path[i].y1-1, path[i+1].x1+1, path[i+1].y1+1);**

**}**

**}**

**draw\_complete = 1;**

**calc\_complete가 0이 아니면 선을 굵기를 5로 하고, 그림의 색깔을 uniqueColor\_r,g,b를 이용하여 설정한다. 그리고 path의 i번째 좌표부터 path의 i+1번째 좌표까지 라인을 그리는 동작을 반복한다. 경로를 모두 그렸으면 draw\_complete에 1을 저장한다.**

**if( key == 'e'){**

**if(draw\_flag)**

**waterfall\_start\_flag = 0;**

**}**

**사용자가 e를 누르면 draw\_flag가 0이 아니면 waterfall\_start\_flag를 0으로 설정한다.**

**도표이(가) 표시된 사진

자동 생성된 설명**

**도표이(가) 표시된 사진

자동 생성된 설명**

**도표이(가) 표시된 사진

자동 생성된 설명**

**도표이(가) 표시된 사진

자동 생성된 설명**

**도표이(가) 표시된 사진

자동 생성된 설명**

**도표이(가) 표시된 사진

자동 생성된 설명**