import unlekker.mb2.geo.\*;

import unlekker.mb2.util.\*;

import unlekker.data.\*;

import unlekker.mb2.externals.\*;

import ec.util.\*;

import fingertracker.\*;

import SimpleOpenNI.\*;

import processing.serial.\*;

String filename="aapl2008"; // stl 저장 파일 이름

UNav3D nav; // 3차원 뷰

FingerTracker fingers; //손가락 트래커 라이브러리 사용

SimpleOpenNI kinect; //Kinect를 위한 라이브러리 사용

int reset = 5; //객체 5번 쌓고 리셋

int threshold = 625; //손가락 트래커 거리 문턱값

int col\_count, finger\_count, ave\_col, time\_count=0, reset\_count=0; //카운팅 변수 정의

PImage currentFrame; // 현재 이미지 저장 정의

color trackColor; // 컬러 영상에서 카운팅할 컬러 값 정의

Serial commPort ; // 씨리얼 값 받는 포트 정의

String row\_data; //불필요

char HEADER = 'H'; // 씨리얼 입력 값 시작점 정의

float y, h;//불필요

int tempC, tempF, RH, UV, Mic, light; // 센서 값 저장 변수 정의

int temp\_init, RH\_init, UV\_init, light\_init;// 초기 센서 값 저장 변수 정의

//센서값 비교 배열 정의

int[] D1 = new int[40];

int[] D2 = new int[40];

int[] D3 = new int[40];

int[] D4 = new int[40];

int[] D5 = new int[40];

int[] D6 = new int[40];

int[] D7 = new int[40];

//셋팅 함수

public void setup() {

size(800, 800, P3D);

UMB.setPApplet(this); //mk2 라이브러리 사용

nav=new UNav3D(); //3차원 뷰 사용

nav.rot.set(radians(90), radians(10), radians(10)); //3차원 뷰 시작점 회전 정의

nav.trans.set(50, 50, 400); // 3차원 뷰 움직임 정의

kinect = new SimpleOpenNI(this); // Kinect 사용 정의

kinect.enableDepth(); //깊이 값 사용

kinect.setMirror(true); //스켈레톤 연결 안되게

fingers = new FingerTracker(this, 640, 480); //손가락 트래커 사용

kinect.enableRGB(); // 컬러 사용

trackColor = color (120,160,60); // 컬러 카운팅 값 정의

smooth (); // 스무딩

fingers.setMeltFactor(100); // 손가락 세팅

// handPositions = new ArrayList();

currentFrame = createImage (640,480, RGB); //현재 컬러 영상 크기 정의

commPort = new Serial(this, "com8", 9600); // 씨리얼 포트 값

//size(kinect.depthWidth()\*2, kinect.depthHeight());

//stroke(255,0,0);

//strokeWeight(2);

// build();

// if(time\_count >60)

// {

build(finger\_count, ave\_col, temp\_init, UV\_init, light\_init, Mic, RH\_init); //센서 값 받아서 객체 생성하는 함수

// time\_count = 6;

// }

// colorMesh();

/\*if(time\_count ==7)

{

lstart(ave\_col);

}\*/

}

public void draw() {

background(0); // 배경

kinect.update();// 키넥트 갱신

PImage depthImage = kinect.depthImage(); // 깊이 이미지 받기

// image(depthImage,0,0);

currentFrame = kinect.rgbImage(); // 컬러 이미지 받기

// image(currentFrame,640,0);

// strokeWeight(10);

float worldRecord = 500;

fingers.setThreshold(threshold);// 손가락 값 세팅

// XY coordinate of closest color

int closestX = 0;

int closestY = 0;

col\_count =0;

// Begin loop to walk through every pixel

for (int x = 0; x < currentFrame.width; x ++ ) {

for (int y = 0; y < currentFrame.height; y ++ ) {

int loc = x + y\*currentFrame.width;

// What is current color

color currentColor = currentFrame.pixels[loc];

float r1 = red(currentColor);

float g1 = green(currentColor);

float b1 = blue(currentColor);

float r2 = red(trackColor);

float g2 = green(trackColor);

float b2 = blue(trackColor);

// Using euclidean distance to compare colors

float d = dist(r1,g1,b1,r2,g2,b2); // We are using the dist( ) function to compare the current color with the color we are tracking.

// If current color is more similar to tracked color than

// closest color, save current location and current difference

if (d < worldRecord) {

col\_count++;

worldRecord = d;

closestX = x;

closestY = y;

}

}

}

// We only consider the color found if its color distance is less than 10.

// This threshold of 10 is arbitrary and you can adjust this number depending on how accurate you require the tracking to be.

/\*if (worldRecord < 100) {

// Draw a circle at the tracked pixel

fill(trackColor);

strokeWeight(4.0);

stroke(0);

ellipse(closestX+640,closestY,16,16);

}\*/

int[] depthMap = kinect.depthMap();

fingers.update(depthMap);

stroke(0,255,0);

for (int i = 0; i < fingers.getNumFingers(); i++) {

PVector position = fingers.getFinger(i);

// ellipse(position.x - 5, position.y -5, 10, 10);

}

finger\_count = fingers.getNumFingers();

// println("Thres="+threshold);

//println("y="+y);

fill(255,0,0);

//센서 값 차이 구하기

for(int j = 39; j!=0; j--){

D1[j] = D1[j-1];

D2[j] = D2[j-1];

D3[j] = D3[j-1];

D4[j] = D4[j-1];

D5[j] = D5[j-1];

D6[j] = D6[j-1];

D7[j] = D7[j-1];

}

D1[0] = finger\_count;

D2[0] = ave\_col;

D3[0] = tempC;

D4[0] = UV;

D5[0] = Mic;

D6[0] = RH;

D7[0] = light;

//센서 값 화면에 출력

for(int j = 0; j<40; j++){

fill(255,0,0);

text(D1[j], 10, 20\*(j+1));

fill(255,0,0);

text(D2[j], 50, 20\*(j+1));

fill(255,0,0);

text(D3[j], 90, 20\*(j+1));

fill(255,0,0);

text(D4[j], 130, 20\*(j+1));

fill(255,0,0);

text(D5[j], 170, 20\*(j+1));

fill(255,0,0);

text(D6[j], 210, 20\*(j+1));

fill(255,0,0);

text(D7[j], 250, 20\*(j+1));

}

if(time\_count ==1800)

{

grow(finger\_count, ave\_col, tempC, UV, light, Mic, RH); // 객체를 쌓는 함수

time\_count = 11;

reset\_count ++;

}

if(reset\_count ==reset)

{

build(finger\_count, ave\_col, temp\_init, UV\_init, light\_init, Mic, RH\_init); //리셋 값에 의해 초기화

reset\_count =0;

}

// if tiler exists, see if we are done tiling. if so, set tiler to null.

/\* if (tiler!=null && tiler.done) {

tiler=null;

} else if(tiler==null) {

// the credit text should only be drawn when not tiling

drawCredit();

}\*/

translate(width/2, height/2);

lights();

nav.doTransforms();

noStroke();

fill(255);

geo.draw();

// geo1.draw();

}

//씨리얼 값 처리 함수

void serialEvent(Serial p) {

// get message till line break (ASCII > 13)

String message = commPort.readStringUntil('\n');

if (message != null) {

// try catch function because of possible garbage in received data

try {

//print(message);

String[] elements = message.split(",");//splitTokens(message);

//씨리얼 데이터가 ‘H’로 시작하고 10초 지났으면 센서 값 차이 가져오기

if(elements[0].charAt(0) == HEADER && elements.length >= 1 && time\_count >10) // check validity

{

RH = int(elements[1])-RH\_init;

tempC = int(elements[2])-temp\_init;

UV = int(elements[4])-UV\_init;

light = int(elements[3])-light\_init;

Mic = int(elements[5]);

}

//씨리얼 데이터가 ‘H’로 시작하고 10초 경과했으면 초기 센서 값 가져오기 그리고 초기 객체 생성하기

else if(elements[0].charAt(0) == HEADER && elements.length >= 1 && time\_count ==10) // check validity

{

RH\_init = int(elements[1]);

temp\_init = int(elements[2]);

UV\_init = int(elements[4]);

light\_init = int(elements[3]);

ave\_col = (ave\_col+col\_count)/2;

build(finger\_count, ave\_col, temp\_init, UV\_init, light\_init, Mic, RH\_init);

}

센서 값 차이, 시간, 컬러, 손가락 값 출력하기

print("tempC: "+tempC);

print("UV: "+UV);

print("light: "+light);

print("Mic: "+Mic);

println("RH: "+RH);

delay(1000);

time\_count++;

print("color: "+ave\_col);

print("finger: "+finger\_count);

println("time: "+time\_count);

}

catch (Exception e) {

}

}

}

//마우스 클릭시 쌓기

void mousePressed() {

grow(finger\_count, ave\_col, tempC, UV, light, Mic, RH);

}

-----build 클래스

//객체 생성하기 클래스

ArrayList<UVertexList> vvl;

UVertexList vl, vl2;

UGeo geo; //객체 정의

UGeo geo1; //객체2 정의

UGeo geo2; //객체3 정의

int ave\_leaf =3; // leaf count pram

float x\_cod, y\_cod;

int check, vol;

//움직임, 회전 경로 정의

float[] posX = new float[10];

float[] posY = new float[10];

float[] posZ = new float[10];

float[] rosX = new float[10];

float[] rosY = new float[10];

float[] rosZ = new float[10];

void build(int depth, int count, int tempC, int UV, int light, int Mic, int RH) {

int leaf\_count = count/ave\_leaf; //컬러 카운트에 지정된 값으로 나눠서 객체 총 수 결정

if(leaf\_count >10) leaf\_count = 10; // 최대 10개 까지

geo=UGeoGenerator.geodesicSphere(15, 2); //15mm 2타입 구 객체 생성

posX[0] = 0; // 최초 객체 위치 x

posY[0] = 0; // 최초 객체 위치 y

posZ[0] = 7.5; // 최초 객체 위치 z

geo.translate(0 , 0 , posZ[0] );

geo1= UGeoGenerator.meshBox(150,150,3,2) ; // 바닥 객체2에 생성

geo.add(geo1); 출력 객체에 더하기

for(int i=0; i<leaf\_count-1; i++){

if(i==0){

posX[1] = -tempC; //2번째 객체는 온도에 따라 위치 정의

posY[1] = -tempC;

}else if(i==1){

posX[2] = UV/10; // UV 값에 따라 위치 정의

posY[2] = UV/10;

}

else if(i==2){

posX[3] = light/10; //밝기 값에 따라 위치 정의

posY[3] = light/10;

}else if(i==3){

posX[4] = -RH; // 습도에 따라 위치 정의

posY[4] = -RH;

}

else{

posX[i+1] =random(-50,50); // 나머지 랜덤 위치

posY[i+1] = random(-50,50);}

posZ[i+1] =7.5;

geo1=UGeoGenerator.geodesicSphere(15, 2);// 각 위치마다 구 객체 생성

geo1.translate(posX[i+1] , posY[i+1],posZ[i+1]); // 생성 위치 저장

geo.add(geo1); // 객체 통합

}

for(int i=0; i<leaf\_count; i++)

{

rosX[i] = 0; // 초기 회전 리셋

}

geo.add(geo1);

geo.regenerateFaceData();;

geo.writeSTL(sketchPath+"/"+UFile.noExt(filename)+".stl"); // stl파일로 출력

}

int cnt=0;

void deform() { //deformation 사용 안함

UFace f=UMB.rnd(geo.getF());

UVertex fv=f.centroid().copy().mult(0.1);

fv.add(frameCount/1000,0,0);

float force=sin(noise(fv.x,fv.y,0)\*PI);

force\*=random(100)>80 ? -random(10,30) : random(10,30);

// f.translate(random(-10,10) ,0 ,random(-10,10) );

f.translate(new UVertex(f.normal().copy().mult(force)));

// geo.regenerateFaceData(geo.getF().indexOf(f));

if(cnt>0 && (cnt++)%30==0) USubdivision.subdivide(geo,UMB.SUBDIVMIDEDGES);

}

//객체 쌓기 함수

void grow(int depth, int count, int tempC, int UV, int light, int Mic, int RH) {

int leaf\_count = count/ave\_leaf; // 컬러에 따라 객체 수 결정

if(leaf\_count >10) leaf\_count = 10; // 10이 최고 객체수

int ch1, ch2;

for(int i=0; i<leaf\_count; i++){

check = (int)random(0, 9); //랜덤 객체 종류 선택

vol = (int)random(15, 20); //객체 크기 랜덤 15~20mm

if(Mic < 500) //마이크 값이 500 보다 작으면 7:1:1:1 비율

{ ch1 = 7; ch2 = 1;}

else {ch1 = 4; ch2 = 2;} // 크면 4:2:2:2 비율

posX[i] = posX[i] + random(-10, 10); //가로 세로 랜덤 -10~10mm 이동

posY[i] = posY[i] + random(-10, 10);

posZ[i] = posZ[i]+10;

rosX[i] = rosX[i] + random(0.01, 0.01+depth/500); // 랜덤하게 각도 변화 손가락 수만큼 증가

//객체 종류

if(check <ch1)

{

geo1=UGeoGenerator.geodesicSphere(vol, 2); // 구

geo1.translate(posX[i] ,posY[i] , posZ[i]);

geo1.rotX(rosX[i]);

geo.add(geo1);

}

else if(check >=ch1 && check < ch1+ch2 )

{

geo1=UGeoGenerator.geodesicSphere(vol, 2); // 구 + 가운데 박스

geo1.translate(posX[i] ,posY[i] , posZ[i]);

geo1.rotX(rosX[i]);

geo.add(geo1);

geo1 = UGeoGenerator.meshBox(vol,vol,vol/2,2) ;

geo1.translate(posX[i] ,posY[i] , posZ[i]);

geo1.rotX(rosX[i]);

geo.add(geo1);

}

else if(check >=ch1+ch2 && check < ch1+ch2\*2 )

{

geo1=UGeoGenerator.meshBox(vol,vol,vol,2) ; 박스

geo1.translate(posX[i] ,posY[i] , posZ[i]);

geo1.rotX(rosX[i]);

geo.add(geo1);

}

else if(check >=ch1+ch2\*2 && check < ch1+ch2\*3 )

{

geo1=UGeoGenerator.meshBox(vol,vol,vol,2) ; 박스 회전

geo1.rotX(0.5);

geo1.rotY(0.5);

geo1.rotZ(0.5);

geo1.translate(posX[i] ,posY[i] , posZ[i]);

geo1.rotX(rosX[i]);

geo.add(geo1);

}

}

geo.writeSTL(sketchPath+"/"+UFile.noExt(filename)+".stl"); stl 파일에 저장

}