Coding2-Final -Project

Kexin_Mei_20013506

My Github Link : **https://github.com/orionmel/Coding2_Final_Artwork_Advanced_Framework**

My Video Link : https://vimeo.com/689267424

Bonus : My Face web

My Video Link : https://vimeo.com/689315352

Say Your Love

Design Background：

The National Covid Memorial Wall, located outside St Thomas' Hospital, is a public mural built to commemorate a loved one lost in the UK due to covid-19, and each painted heart represents a unique way of commemorating. And the purpose of this wall is so that people do not forget the deceased.
I used this wall as the background, designed the interactive art work "Say Your Love", although covid-19 brings people the pain of losing loved ones, but also let us know how to cherish the people around us, we need to sincerely say your love to the people around us, cherish the happiness in front of us

The description of my code：

1. The main visual interactive code description：

(1) A 3D motion plane created by the Perlin noise in openframeworks

· In the [ofApp.hpp] page , create mesh and main camera

```
class ofApp : public ofBaseApp{

    public:
      void setup();
      void update();
      void draw();

      ofMesh mainMesh;
      ofEasyCam mainCam;

      int width, height;
      bool b_messyMesh, b_perlinMesh, b_drawWireFrame, b_marchMesh;
      float perlinRange, perlinHeight;
};
```

· In the [ofApp.cpp] page

basic setting & make points within the grid

```
void ofApp::setup(){
/// basic setting
width = 100;
height = 100;
b_drawWireFrame = true;
perlinRange =1.0;
perlinHeight = 5.0;
ofBackground(0);
ofSetColor(255);
mainCam.setPosition(0,0, 100);
mainCam.orbit(0,-80, -20);

// Here we make points within the grid
for (int y = 0; y < height; y++){
           for (int x = 0; x<width; x++){
               mainMesh.addVertex( ofPoint(x - width /2 , y - height / 2 ,10)); // mesh index = x + y*width
               mainMesh.addColor(ofFloatColor(255,255,255));//主要的颜色

           }
      }


      for (int y = 0; y<height-1; y++){
           for (int x=0; x<width-1; x++){
               mainMesh.addIndex(x+y*width);               // 0
               mainMesh.addIndex((x+1)+y*width);           // 1
               mainMesh.addIndex(x+(y+1)*width);           // 10
```

```
                mainMesh.addIndex((x+1)+y*width);           // 1
                mainMesh.addIndex((x+1)+(y+1)*width);       // 11
                mainMesh.addIndex(x+(y+1)*width);           // 10
            }
        }
}
```

Move the grid forward

```
void ofApp::update(){
    ofColor newColor;
    ofVec3f newPosition, oldPosition;

     for (int y = height; y>0; y--){
         for (int  x= width; x>0; x--){
        oldPosition = mainMesh.getVertex( x + ( y  * width));
        newPosition = mainMesh.getVertex(x + ( (y - 1) * width) );
        oldPosition.z = newPosition.z;
        mainMesh.setVertex( x + ( y * width ), oldPosition); //更新顶点的位置
        newColor.setHsb(150 ,  255 - ofMap( newPosition.z, 0,  perlinHeight, 0 ,  255), 255);
        mainMesh.setColor(x + ( y * width ), newColor);

          }
     }
     for (int x=0; x<width; x++){
     newPosition = mainMesh.getVertex(x); // 获取此顶点的当前 x，y，z 位置
     newPosition.z = ofRandom(-1, 1); // 将其的 z 值设置为新的随机数
     mainMesh.setVertex(x, newPosition); // 使用新坐标更新顶点的位置
      }

}
```

Draw the grid

```
void ofApp::draw(){

        glPointSize(3);
        mainCam.begin();

        if (b_drawWireFrame){
            //选择将网格绘制为线框
            mainMesh.drawWireframe();
        } else {
            //选择将网格绘制为点云
            mainMesh.drawVertices();
        }

        mainCam.end();

 }
```

(2) Drawing the [heart] in the middle

· **In the [ofApp.hpp] page**

```
class ofApp : public ofBaseApp{
    glm::vec2 make_point(float theta);
    vector<ofColor> base_color_list;
    float theta_seed, scale_seed;
    vector<pair<float, float>> param_list;
    vector<vector<glm::vec2>> log_list;
    vector<ofColor> color_list;
    vector<float> life_list;
};
```

· **In the [ofApp.cpp] page**

```
void ofApp::setup(){
    ofEnableBlendMode(ofBlendMode::OF_BLENDMODE_ADD);
    ofNoFill();

    this->theta_seed = ofRandom(1000);
    this->scale_seed = ofRandom(1000);

    ofColor color;
    vector<int> hex_list = { 0xef476f, 0xffd166, 0x06d6a0, 0x118ab2, 0x073b4c };
    for (auto hex : hex_list) {

        color.setHex(hex);
        this->base_color_list.push_back(color);
    }
}
```

create movement effect

```
void ofApp::update(){
        while (this->param_list.size() < 100) {

            auto deg_pair = make_pair(ofRandom(360), ofRandom(3, 20));
            this->param_list.push_back(deg_pair);
            vector<glm::vec2> log;
            this->log_list.push_back(log);
            this->color_list.push_back(this->base_color_list[(int)ofRandom(this->base_color_list.size())]);
            this->life_list.push_back(ofRandom(60, 180));
        }

        for (int i = this->param_list.size() - 1; i >= 0; i--) {

            this->life_list[i] -= 1;
            if (this->life_list[i] < 0) {

                this->param_list.erase(this->param_list.begin() + i);
                this->log_list.erase(this->log_list.begin() + i);
                this->color_list.erase(this->color_list.begin() + i);
                this->life_list.erase(this->life_list.begin() + i);

                continue;
            }

            auto noise_location = this->make_point(this->param_list[i].first);
            auto u_param = ofMap(ofNoise(glm::vec4(noise_location * 0.003, this->param_list[i].second * 0.03, this->theta_seed + ofGetFrame
            auto v_param = ofMap(ofNoise(glm::vec4(noise_location * 0.003, this->param_list[i].second * 0.03, this->scale_seed + ofGetFrame

            this->param_list[i].first += u_param;

          this->param_list[i].second = v_param ;



            this->log_list[i].push_back(this->make_point(this->param_list[i].first) * this->param_list[i].second);
            while (this->log_list[i].size() > 100) {

                this->log_list[i].erase(this->log_list[i].begin());
            }
        }
}
```

drawing heart

```
void ofApp::draw(){
      ofTranslate(ofGetWidth() * 0.5, ofGetHeight() * 0.5);

        for (int i = 0; i < this->param_list.size(); i++) {

            ofSetColor(this->color_list[i]);

            if (this->life_list[i] > 60) {

                ofSetLineWidth(1.5);
            }
            else {
```

```
                ofSetLineWidth(ofMap(this->life_list[i], 0, 60, 0, 1.5));
            }

            ofBeginShape();
            ofVertices(this->log_list[i]);
            ofEndShape();
        }

}

glm::vec2 ofApp::make_point(float theta) {

    float x = 16 * (pow(sin(theta), 3));
    float y = 13 * cos(theta) - 5 * cos(2 * theta) - 2 * cos(3 * theta) - cos(4 * theta);
    return glm::vec2(x, -y);


}
```

(3) Drawing the [heart] in the background

· **In the [ofApp.hpp] page**

```
class ofApp : public ofBaseApp{
      glm::vec2 make_number(float love);
};
```

· **In the [ofApp.cpp] page**

```
void ofApp::draw(){

    /////////////////8❤8❤8❤8❤8❤8❤8❤8❤8❤8❤8❤8❤8❤8❤8❤8❤8❤8❤8❤
    for (int i = 0; i < 50; i++) {

            auto location = glm::vec2(ofRandom(ofGetWidth()), ((int)ofRandom(ofGetHeight() + ofGetFrameNum()) % (ofGetHeight() + 100) - 50)
            auto scale = ofMap(ofNoise(ofRandom(1000), ofGetFrameNum() * 0.005), 0, 1, 1, 3);

            ofPushMatrix();
            ofTranslate(location);
            ofRotate(ofRandom(360));
        ofSetColor(255,ofRandom(100,200),ofRandom(100,150));

            ofBeginShape();
            for (auto deg = 0; deg < 360; deg += 1) {

                ofVertex(this->make_number(deg * DEG_TO_RAD) * scale);
            }
            ofEndShape(true);

            ofPopMatrix();
        }
}

glm::vec2 ofApp::make_number(float love) {

    float x = 16 * (pow(sin(love), 3));
    float y = 13 * cos(love) - 5 * cos(2 * love) - 2 * cos(3 * love) - cos(4 * love);
    return glm::vec2(x, -y);
}
```

(4) Soundinput

· **In the [ofApp.hpp] page**

```
class ofApp : public ofBaseApp{
    void exit();

    void audioIn(float*input,int bufferSize,int nChannels);
    void audioOut(float*buffer,int bufferSize,int nChannels);
```

```
    int sampleRate;
    int bufferSize;

    vector<float>inputBuffer;
};
```

· In the [ofApp.cpp] page

basic setting

```
void ofApp::setup(){
    sampleRate = 48000;
    bufferSize = 512; //256 ,128,64

    inputBuffer.resize(bufferSize*1);

    cout <<(bufferSize*2/(float)sampleRate)*1000.f << endl;
    ofSoundStreamSetup(2,1,sampleRate,bufferSize,4);
};
```

```
void ofApp::draw(){
    ofSetColor(255);
    for(int i = 0 ; i<bufferSize;i++){
        ofCircle(ofGetWidth()/2, ofGetHeight()/2,-(inputBuffer[i]*500) );
    }
};

void ofApp::audioIn(float*input,int bufferSize,int nChannels){
    for(int i = 0;i<bufferSize;i++){
        inputBuffer[i*nChannels+0]=input[i];
    }

}
void ofApp::audioOut(float*buffer,int bufferSize,int nChannels){
    for(int i = 0;i<bufferSize;i++){
        float inputSample;
        float currentSample;

        inputSample = inputBuffer[i];

        currentSample = inputSample;

        buffer[i*nChannels + 0]= currentSample;
        buffer[i*nChannels + 1] = currentSample;
    }
}
void ofApp:: exit(){
    ofSoundStreamClose();
}
```

(5) Background image insertion

I chose this picture

· In the [ofApp.hpp] page

```
class ofApp : public ofBaseApp{
    ofImage img;

};
```

· In the [ofApp.cpp] page

```
void ofApp::setup(){
    img.load("love_wall_test.jpg");
};
```

```
void ofApp::draw(){
    img.draw(0,0);

};
```

(6) Use syphon to connect madmapper and openframeworks

⚠️⚠️⚠️ first to add [ofxSyphon] addon

· In the [ofApp.hpp] page

```
#include "ofxSyphon.h”
class ofApp : public ofBaseApp{
    ofTexture tex;

    ofxSyphonServer mainOutputSyphonServer;
    ofxSyphonServer individualTextureSyphonServer;

    ofxSyphonClient mClient;

};
```

· In the [ofApp.cpp] page

```
void ofApp::setup(){
    mainOutputSyphonServer.setName("Screen Output");
    individualTextureSyphonServer.setName("Texture Output");

    mClient.setup();


    mClient.set("","Simple Server");

    tex.allocate(200, 100, GL_RGBA);

    ofSetFrameRate(60);

};
```

```
void ofApp::draw(){
    mClient.draw(50, 50);

    mainOutputSyphonServer.publishScreen();

    individualTextureSyphonServer.publishTexture(&tex);
}
```

2. The second visual interactive code description：

(1) Drawing the [heart] in the middle

Similar to the code above 👆

(2) The moving line

· In the [ofApp.hpp] page

```
class ofApp : public ofBaseApp{

  public:
    void setup();
    void update();
    void draw();

    ofEasyCam cam;
    int base_radius;
    int number_of_satellite;
    vector<glm::vec3> base_noise_seed_list;
    vector<vector<glm::vec3>> satellite_location_list;
    vector<vector<glm::vec2>> satellite_noise_seed_list;
    vector<glm::vec3> location_list;
    vector<glm::vec3> velocity_list;
    ofMesh face, frame;

};
```

· In the [ofApp.cpp] page

basic setting

```
void ofApp::setup(){

    ofSetColor(255);
    ofEnableDepthTest();

    this->base_radius = 120;
    this->number_of_satellite = 2;
    for (int i = 0; i < 3; i++) {

        this->base_noise_seed_list.push_back(glm::vec3(ofRandom(1000), ofRandom(1000), ofRandom(1000)));

        auto s_location_list = vector<glm::vec3>();
        auto s_noise_seed_list = vector<glm::vec2>();

        for (int k = 0; k < this->number_of_satellite; k++) {

            s_location_list.push_back(glm::normalize(glm::vec3(ofRandom(-1, 1), ofRandom(-1, 1), ofRandom(-1, 1))) * this->base_radius * 1.
            s_noise_seed_list.push_back(glm::vec2(ofRandom(1000), ofRandom(1000)));

        }

        this->satellite_location_list.push_back(s_location_list);
        this->satellite_noise_seed_list.push_back(s_noise_seed_list);
    }

    this->frame.setMode(ofPrimitiveMode::OF_PRIMITIVE_LINES);
};
```

Let line to move

```
void ofApp::update(){
    for (int i = 0; i < this->base_noise_seed_list.size(); i++) {

            auto base_location = glm::vec3(
                ofMap(ofNoise(this->base_noise_seed_list[i].x, (ofGetFrameNum() + 0) * 0.005), 0, 1, -300, 300),
                ofMap(ofNoise(this->base_noise_seed_list[i].y, (ofGetFrameNum() + 0) * 0.005), 0, 1, -300, 300),
                ofMap(ofNoise(this->base_noise_seed_list[i].z, (ofGetFrameNum() + 0) * 0.005), 0, 1, -300, 300)
            );

            for (int k = 0; k < this->number_of_satellite; k++) {

                auto rotation_y = glm::rotate(glm::mat4(), ofMap(ofNoise(this->satellite_noise_seed_list[i][k].x, (ofGetFrameNum() + 0) * 0
                auto rotation_x = glm::rotate(glm::mat4(), ofMap(ofNoise(this->satellite_noise_seed_list[i][k].y, (ofGetFrameNum() + 0) * 0
                auto satellite_location = glm::vec4(this->satellite_location_list[i][k], 0) * rotation_y * rotation_x;

                auto next_base_location = glm::vec3(
```

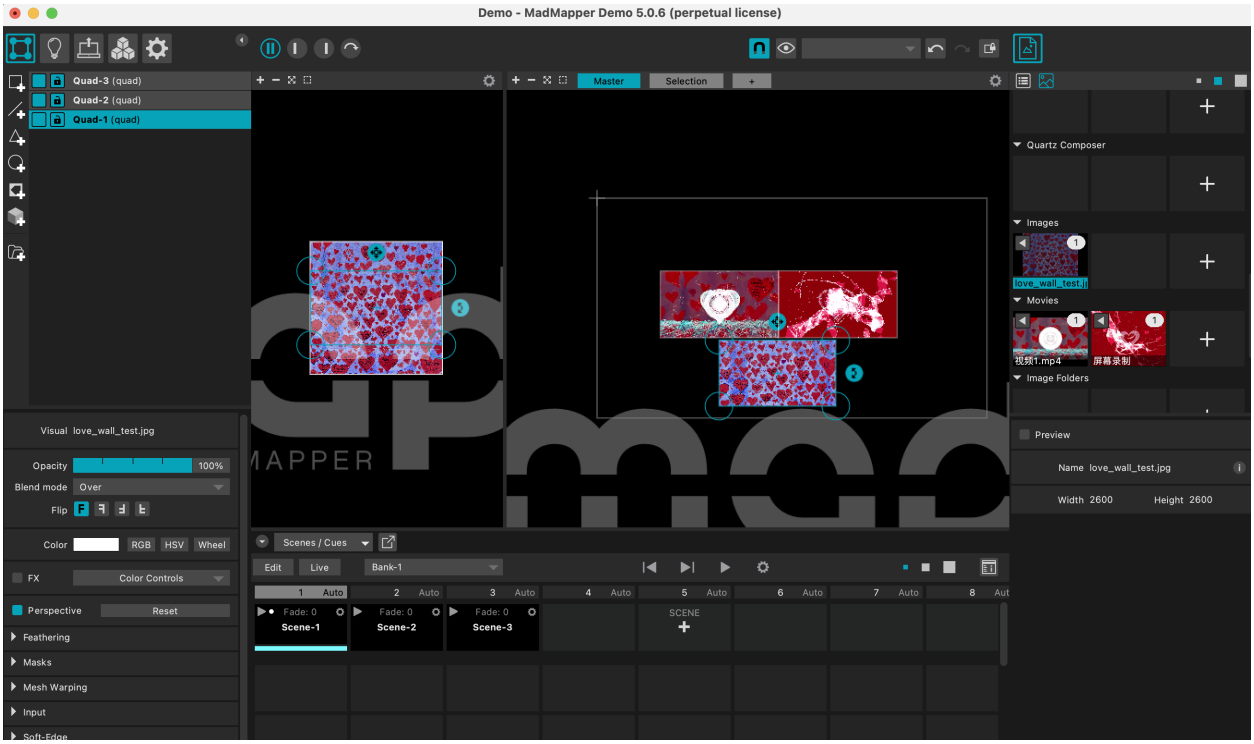
```
                    ofMap(ofNoise(this->base_noise_seed_list[i].x, (ofGetFrameNum() + 1) * 0.005), 0, 1, -300, 300),
                    ofMap(ofNoise(this->base_noise_seed_list[i].y, (ofGetFrameNum() + 1) * 0.005), 0, 1, -300, 300),
                    ofMap(ofNoise(this->base_noise_seed_list[i].z, (ofGetFrameNum() + 1) * 0.005), 0, 1, -300, 300)
                );

                auto next_rotation_y = glm::rotate(glm::mat4(), ofMap(ofNoise(this->satellite_noise_seed_list[i][k].x, (ofGetFrameNum() + 1
                auto next_rotation_x = glm::rotate(glm::mat4(), ofMap(ofNoise(this->satellite_noise_seed_list[i][k].y, (ofGetFrameNum() + 1
                auto next_satellite_location = glm::vec4(this->satellite_location_list[i][k], 0) * rotation_y * rotation_x;

                auto location = base_location + satellite_location;
                auto next_location = next_base_location + next_satellite_location;
                auto distance = location - next_location;

                auto future = glm::vec3(satellite_location) + distance * 80;
                auto random_deg_1 = ofRandom(360);
                auto random_deg_2 = ofRandom(360);
                future += glm::vec3(
                    30 * cos(random_deg_1 * DEG_TO_RAD) * sin(random_deg_2 * DEG_TO_RAD),
                    30 * sin(random_deg_1 * DEG_TO_RAD) * sin(random_deg_2 * DEG_TO_RAD),
                    30 * cos(random_deg_2 * DEG_TO_RAD)
                );
                auto future_distance = future - satellite_location;

                this->frame.addVertex(location);
                this->face.addVertex(location);

                this->frame.addColor(ofColor(255));
                this->face.addColor(ofColor(255, 64));

                this->velocity_list.push_back(glm::normalize(future_distance) * glm::length(distance));
            }
        }

        for (int i = 0; i < this->frame.getNumVertices(); i++) {

            this->frame.setVertex(i, this->frame.getVertex(i) + this->velocity_list[i]);
            this->face.setVertex(i, this->face.getVertex(i) + this->velocity_list[i]);
        }

        for (int i = this->velocity_list.size() - 1; i >= 0; i--) {

            if (glm::length(this->frame.getVertex(i)) > 720) {

                this->velocity_list.erase(this->velocity_list.begin() + i);
                this->frame.removeVertex(i);
                this->face.removeVertex(i);
            }
        }

        this->frame.clearIndices();
        this->face.clearIndices();
        for (int i = 0; i < this->frame.getNumVertices(); i++) {

            vector<int> near_index_list;
            for (int k = i + 1; k < this->frame.getNumVertices(); k++) {

                auto distance = glm::distance(this->frame.getVertex(i), this->frame.getVertex(k));
                if (distance < 50) {

                    this->frame.addIndex(i); this->frame.addIndex(k);
                }

                if (distance < 80) {

                    near_index_list.push_back(k);
                }
            }

            if (near_index_list.size() > 3) {

                for (int k = 0; k < near_index_list.size() - 2; k++) {

                    this->face.addIndex(i);
                    this->face.addIndex(near_index_list[k]);
                    this->face.addIndex(near_index_list[k + 1]);
                }
            }
        }

};
```

Draw the line

```
void ofApp::draw(){
    this->cam.begin();

            for (int i = 0; i < this->frame.getNumVertices(); i++) {

                ofDrawSphere(this->frame.getVertex(i), 2);
            }

            this->frame.drawWireframe();
            this->face.draw();

            this->cam.end();

};
```

3. Projection Mapping

I used syphon, connected openframework to madmapper through the code mentioned above, and then did projection Mapping through my projector, creating a interactive installation



Reference

https://www.youtube.com/watch?v=62WplwZBqLY&t=3s

https://www.youtube.com/watch?v=CetDSun9hfk

https://editor.p5js.org/codingtrain/full/C0kJ-BjYW

https://www.youtube.com/watch?v=-49xDWRPGlA&list=PLNURizt7mHsJ9EasygZJl7M3e-kAOV9Pa&index=11

https://www.youtube.com/watch?v=IiTsE7P-GDs

https://github.com/jocabola/ofxSoundStream

https://www.youtube.com/watch?v=Rbu79hYl1Pk&t=78s

http://syphon.v002.info/#license

https://www.youtube.com/watch?v=wBkvusKre8Q&list=PL4neAtv21WOlqpDzGqbGM_WN2hc5ZaVv7&index=61&t=701s

https://www.youtube.com/watch?v=Lrf3lO9vsCs&list=PLe9qr8GslyxLCyNHgilPmRc6UzgzHfQpM&index=4