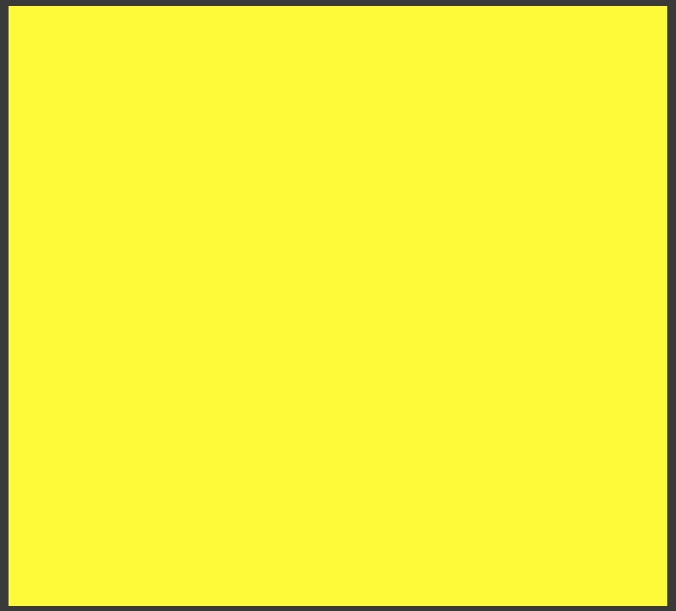
C++

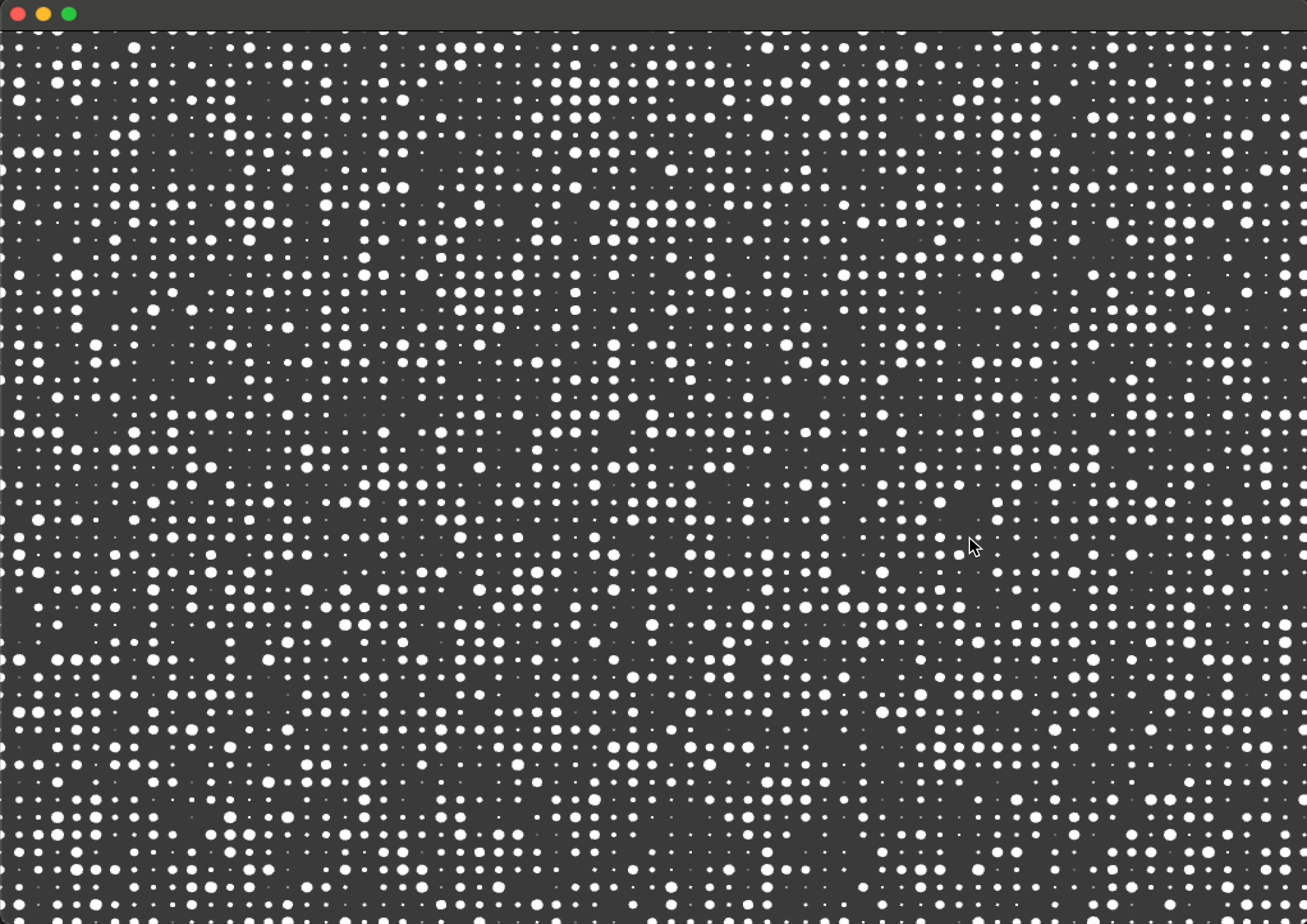


```cpp
#include "ofApp.h"

//--------------------------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------------------------
void ofApp::update(){

}

//--------------------------------------------------------------
void ofApp::draw(){

    //These are the most basic examples. I think they are a good way to kind of just orient yourself with the program, it shows you that your computer is basically like a map and you can plt yourself on it. So when you use a function to draw a rectangle you're giving two plot points for the horizontal and vertical axes and then two dimensional points to say how wide and tall the shape must be. Colours also work in a plot point kind of way. You input three values that make up a colour using red, green and blue values - pretty straightforward stuff, but it was the first thing I learnt so it felt pretty damn cool to actually type something in this kind of format and see a blue square pop out of nowhere, with no error codes and know that I did that. It still kind of bothers me that I have to use the american spelling for the colour function though, but pick your battles I guess.

    ofSetColor(255,250,100);

    ofDrawRectangle(500,300,300,300);

}

```

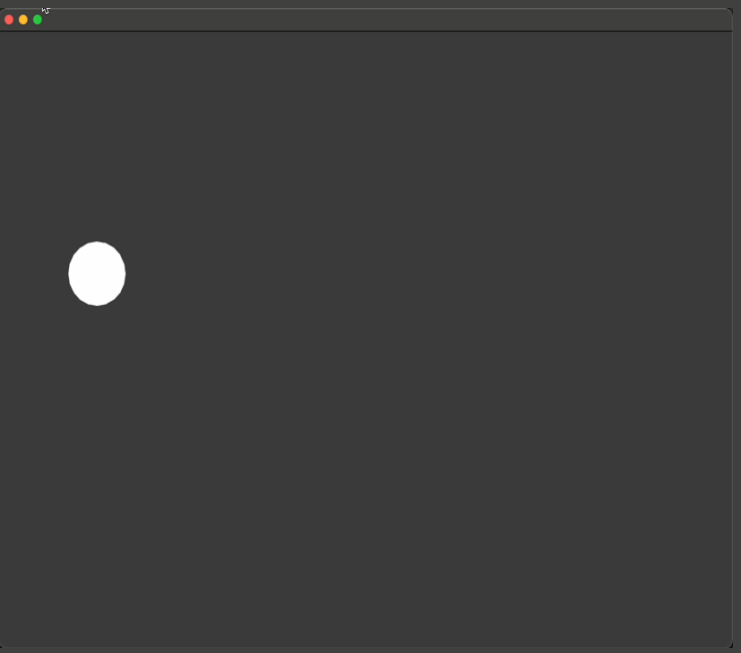
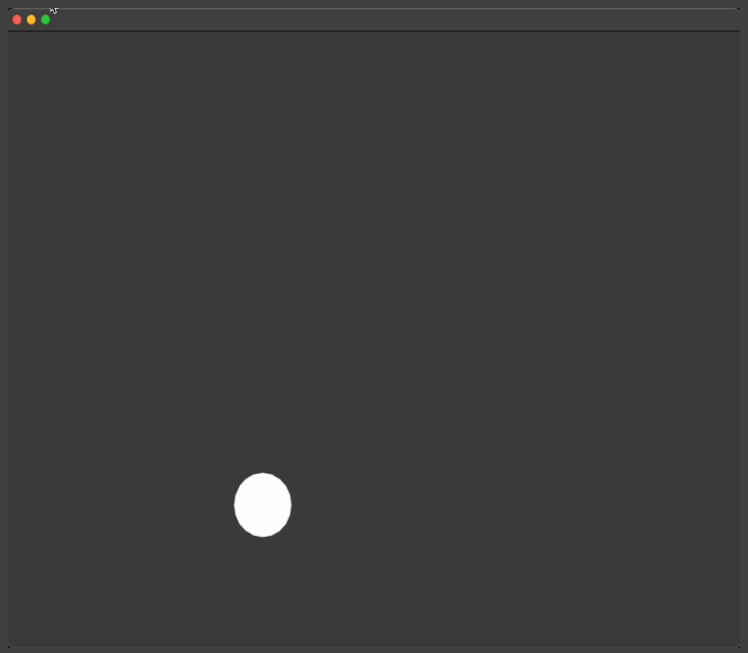
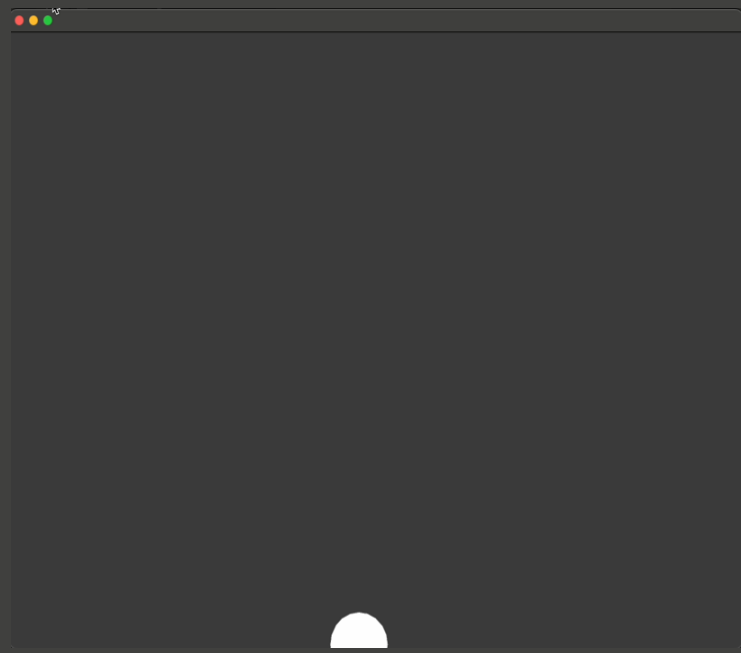
```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{

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

        void keyPressed(int key) override;
        void keyReleased(int key) override;
        void mouseMoved(int x, int y ) override;
        void mouseDragged(int x, int y, int button) override;
        void mousePressed(int x, int y, int button) override;
        void mouseReleased(int x, int y, int button) override;
        void mouseScrolled(int x, int y, float scrollX, float scrollY) override;
        void mouseEntered(int x, int y) override;
        void mouseExited(int x, int y) override;
        void windowResized(int w, int h) override;
        void dragEvent(ofDragInfo dragInfo) override;
        void gotMessage(ofMessage msg) override;

    // Things start to get a little more interesting here. First off, I'm in a completely different file - I mean we're still in
        Xcode, but this now I'm using the .h file to declare variables before I initialise them in the main .cpp file. I still
        only understand about half of what that means or why I need to do it, but I'm hoping that as I work with this stuff more
        and more it will become clearer, for now I'm just following steps...
    int x;
    int y;
    int w;
    int h;
    bool state;


};
```

```cpp
#include "ofApp.h"

//--------------------------------------------------------------
void ofApp::setup(){

    //Back to the .cpp editor, here are those variables again from before, I guess now that I've established them in the header file I can use them in this one? I'm also starting to use conditions which I immediately enjoyed. It's quite a simple format of yes/no, true/false outcomes and contigencies. So if this happens the program should do this, but if it doesn't happen the program should do that instead.

    x=200;
    y=300;
    w=250;
    h=430;
    state=false;

}

//--------------------------------------------------------------
void ofApp::update(){

    //so here are those condition statements, it looks more complicated than it is (or maybe it looks less complicated than it is? I don't know it took me a minute to wrap my head around this one), but basically I'm trying to tell the program that I'm making a rectangular button and when my mouse hovers over it, it should be green and when it's not over the button, it should be red - simple right? Well the function below basically tells the program exactly that, but instead of fixed positions for the x and y plot points I'm using dynamic functions that will follow my mouse around the screen and actively update the position coordinates. Oh and I'm establishing what is true and what isn't, so in this case true means that my mouse is over the rectangle and false means that it's not. Of course the position of the mouse isn't objectively true or false, but the kind of yes/no structure words just become signifiers for either/or options.
    if(ofGetMouseX()>x && ofGetMouseX()<x+w && ofGetMouseY()>y && ofGetMouseY()<y+h){
        state=true;
    }
    else{
        state=false;
    }
}
```

```
//--------------------------------------------------------------
void ofApp::draw(){

    //basic stuff here, just drawing rectangles and setting colours.

    if(state==true){
        ofSetColor(0,255,0);
    }
    else{
        ofSetColor(255,0,0);
    }

    ofDrawRectangle(x,y,w,h);
}

//--------------------------------------------------------------
void ofApp::exit(){

}

//--------------------------------------------------------------
void ofApp::keyPressed(int key){

}

//--------------------------------------------------------------
void ofApp::keyReleased(int key){

}

//--------------------------------------------------------------
void ofApp::mouseMoved(int x, int y ){

}
```



```
2
3  //--------------------------------------------------------------
4  void ofApp::setup(){
5
6      //Now I've started working with some loops - essentially repeating images or animations. In between these last two
           examples I started working with the sin and cos functions (yes the same ones I learnt in math and confidently told my
           highschool math teacher I would never need or use again) that also work well with animations because they fluctuate
           between 1 and -1, but fluctuate up and down contiuously. The program crashed though and I thoought this example of
           animations gives about the same effect and understanding so I was lazy and didn't recreate those demos - also I
           haven't completely figured out what made them crash in the first place and I'm a little scared to go back and do it
           again. Anyways, back to loops, they're basically a sequence of instructions that allow me to repeat a block of code
           and manipulate large sets of data. This one isn't that fancy yet though and just uses the ofRandom function to make a
           grid of circles randomly change their radius size. This doesn't sound very interesting, but the result is kind of
           disco-esque which I did like.
7
8      cout<<"before the loop"<<endl;
9
10     for(int i=0; i<10; i++){
11
12         cout<<"i:"<<i<<endl;
13
14     }
15
16     cout<<"after the loop"<<endl;
17 }
18
19
20 //--------------------------------------------------------------
21 void ofApp::update(){
22
23 }
24
25 //--------------------------------------------------------------
26 void ofApp::draw(){
27
28     for(int i=0;i<150;i++) {
29
30         for(int j=0;j<100;j++) {
31
32             ofDrawCircle(i*15,j*15,ofRandom(0,5));
33         }
34
35
36     }
37 }
38
```

```
//
void ofApp::setup(){

    // This animation works with a few of the same elements as the last demo. It seems like a more basic animation, but the
        shape that is moving is moving positions across the screen - it's pretty similar to that old dvd screen saver, that
        one where the logo bounced around the screen and you would sit and wait for it to perfectly hit the corner of the tv
        screen and then bounce away again. So I'm using the ofRandom function again so that the path of the ball is
        consistently different. That's whats happening in this section.

    pos.x=ofRandom(0,ofGetWidth());
    pos.y=ofRandom(0,ofGetHeight());

    vel.x=ofRandom(-10,10);
    vel.y=ofRandom(-10,10);


}

//--------------------------------------------------------------
void ofApp::update(){

    // Over here I'm looking at the speed of the ball and making sure that when it reaches the edge of the screen that it
        bounces and returns rather than just disappears off screen into oblivion. In my earlier attempts at this animation I
        forgot to include the conditions that make sure it returns and it just floated off, which kind of made me wonder how
        far it travels, because even when it was off-screen the program kept running so even though I couldn't see it the app
        was still tracking the ball somewhere.

    ofSetFrameRate(30);

    pos+=vel;

    if ((pos.x > ofGetWidth())||(pos.x<0)){
        vel.x*=-1;
    }

    if ((pos.y>ofGetHeight())||(pos.y<0)){
        vel.y*=-1;
    }
}
```

```
//
//  ball.h
//  bouncingBallClassDemo
//
//  Created by Emma van Schalkwyk on 2024/03/11.
//
// I tried to compile and the build failed - apparantly it couldn't find the ball.h file... Not sure exactly why this is, but I think I made an .hpp file instead of an .h file so I deleted the original one, which I probably shouldn't have done because I forgot to copy all the code I already set up and have to do it again now, but it's fine, this can be practice to do it from my notes rather than the course. Anyways, I'm going to finish setting up and running this again, but if it doesn't work after this I might lose my shit a little

//Update: ran it again, didn't work, actually created more errors somehow. I don't know what I'm doing wrong here.

//Update: I DID IT!!! I deleted all of my classes, created new classes, re-entered all of the code manually from my notes ran it three times and it finally worked. In retrospect I probably could have just created a new .h file, copied and pasted the code from the .hpp file and then deleted the .hpp file, but retrospect is a bitch and I only know this now because I did it wrong in the first place. Anyways, the app looks basically the same as the first time I did this demo, all that's different is that I used classes this time, so yeah I'm really glad I totally didn't just waste 3 hours trying to figure this out :')

#ifndef ball_h
#define ball_h

#include <stdio.h>


#endif /* ball_h */
#include "ofMain.h"
class ball{
public:
    ball();
    void setup(ofVec2f initialPos,ofVec2f initialVel,float rad);
    void update();
    void draw();

    ofVec2f pos;
    ofVec2f vel;
    float radius;
```

ball | ball | ofApp | ofApp

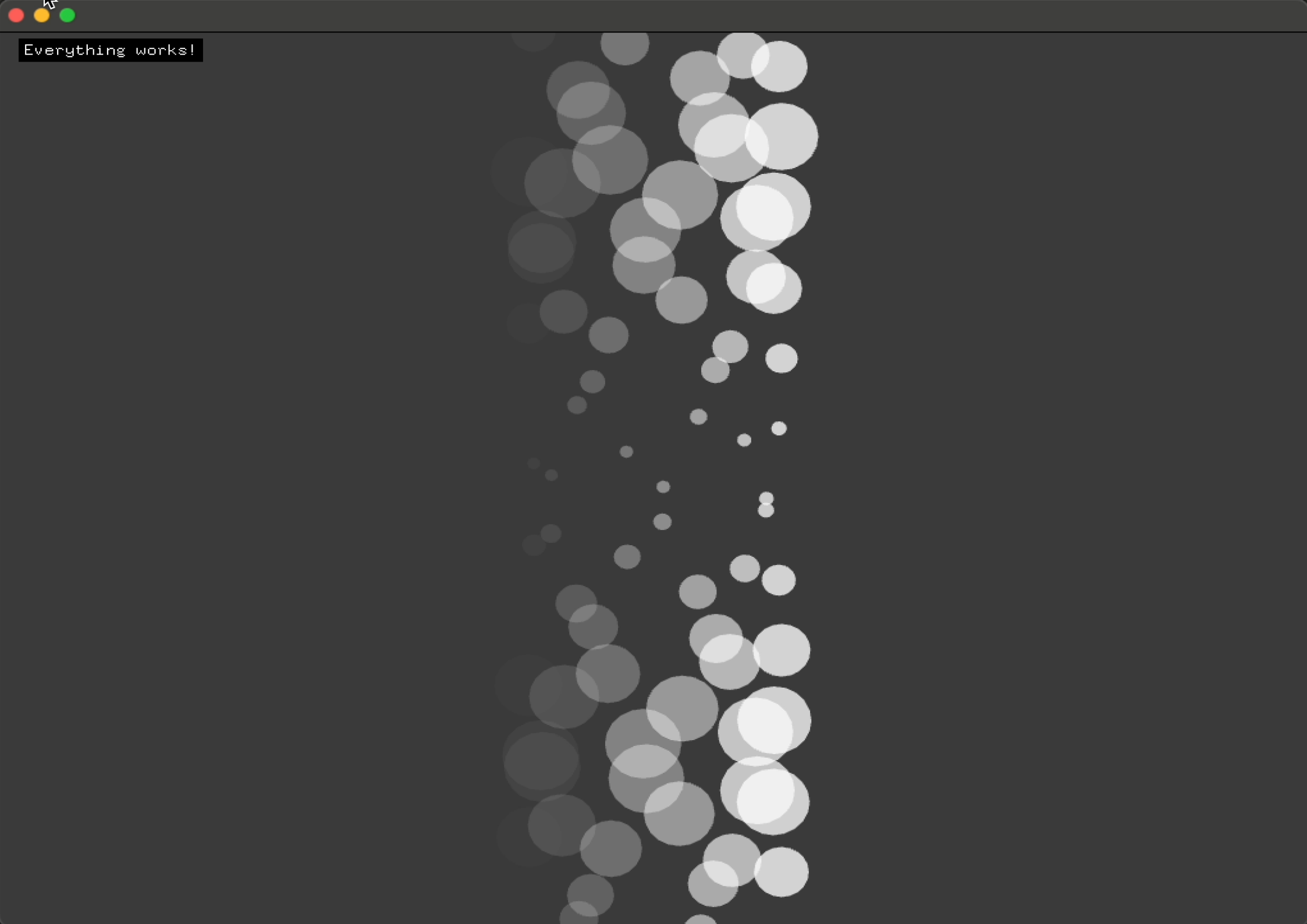
bouncingBallClassDemo 〉 src 〉 ofApp 〉 ofApp::setup()

```
#include "ofApp.h"

//--------------------------------------------------------------
void ofApp::setup(){

    for (int i=0; i<200;i++) {
        fastBall
            [i]
            .setup(ofVec2f(ofRandom(0,ofGetWidth()),ofRandom(0,ofGetHeight())),ofVec2f(ofRandom(-2,2),ofRandom(-2,2)),ofRandom
            (20,100));
    }


}

//--------------------------------------------------------------
void ofApp::update(){
    for (int i=0; i<200;i++) {
        fastBall[i].update();
    }

}

//--------------------------------------------------------------
void ofApp::draw(){
    for (int i=0;i<200;i++) {
        fastBall[i].draw();

    }

}

//--------------------------------------------------------------
void ofApp::exit(){

}
```

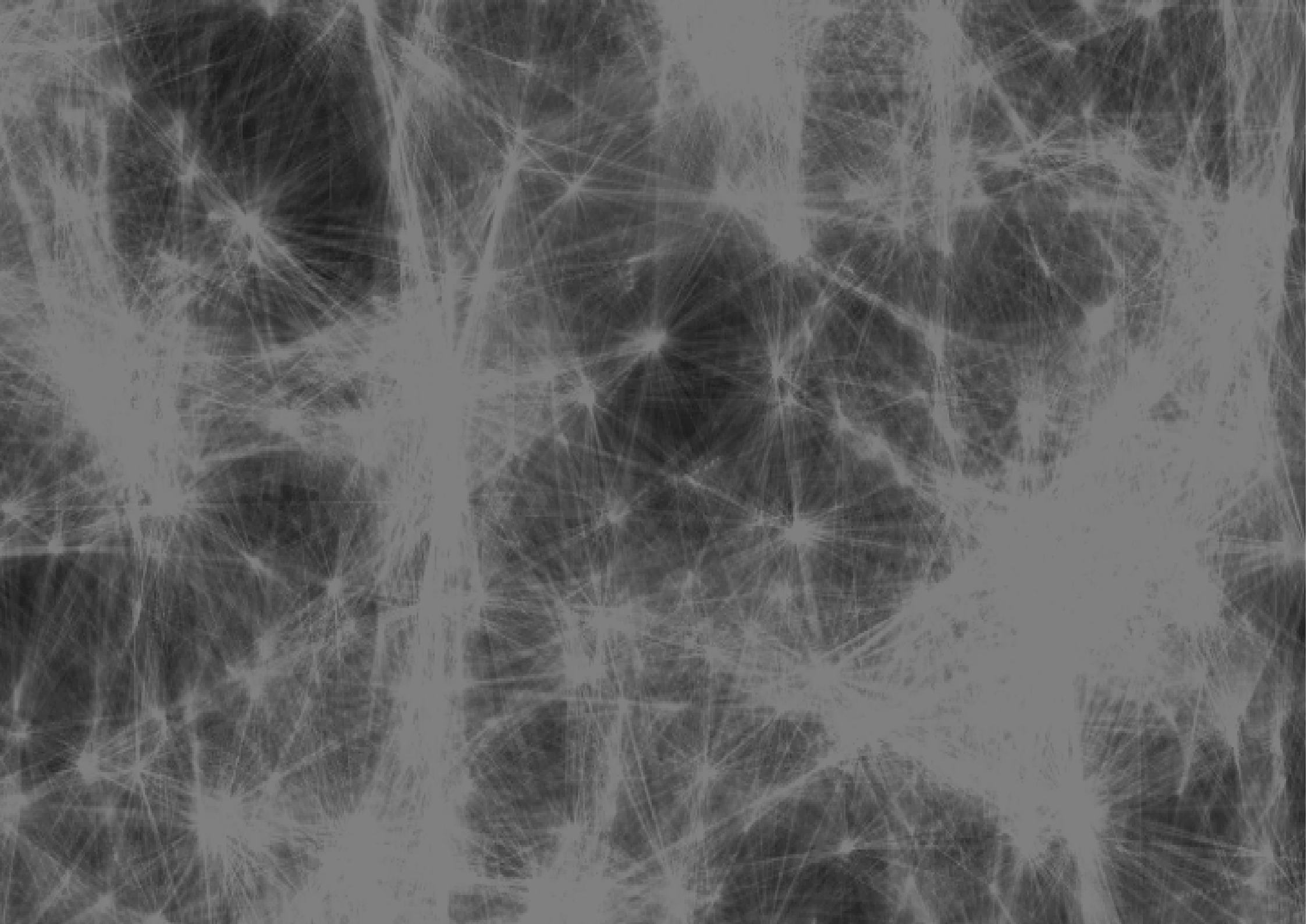
ball | ball | ofApp | ofApp

bouncingBallClassDemo › src › ofApp › ofApp

```
#pragma once

#include "ofMain.h"
#include "ball.h"

class ofApp : public ofBaseApp{

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

        void keyPressed(int key) override;
        void keyReleased(int key) override;
        void mouseMoved(int x, int y ) override;
        void mouseDragged(int x, int y, int button) override;
        void mousePressed(int x, int y, int button) override;
        void mouseReleased(int x, int y, int button) override;
        void mouseScrolled(int x, int y, float scrollX, float scrollY) override;
        void mouseEntered(int x, int y) override;
        void mouseExited(int x, int y) override;
        void windowResized(int w, int h) override;
        void dragEvent(ofDragInfo dragInfo) override;
        void gotMessage(ofMessage msg) override;

    ball fastBall[200];

};

```

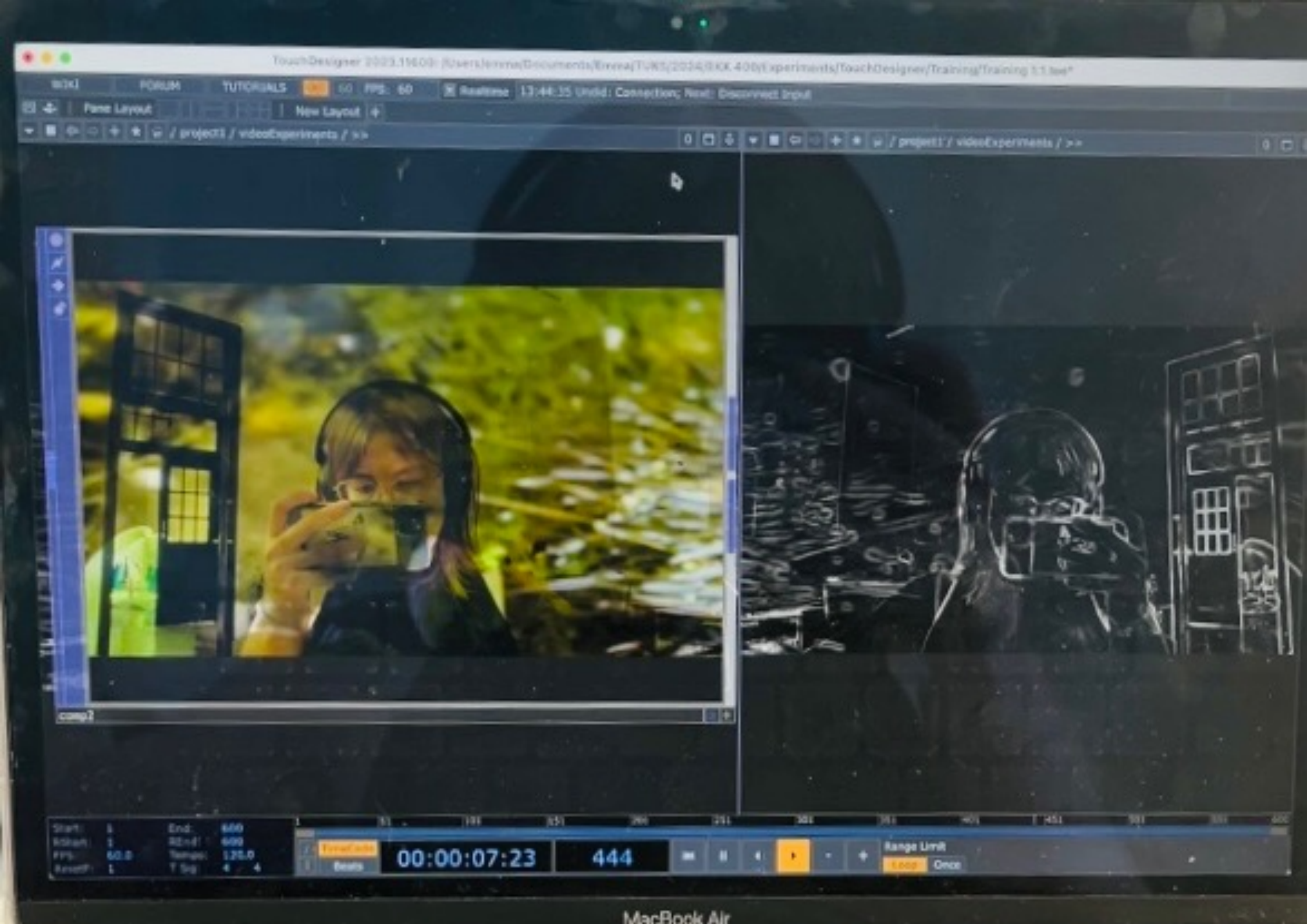

Everything works!

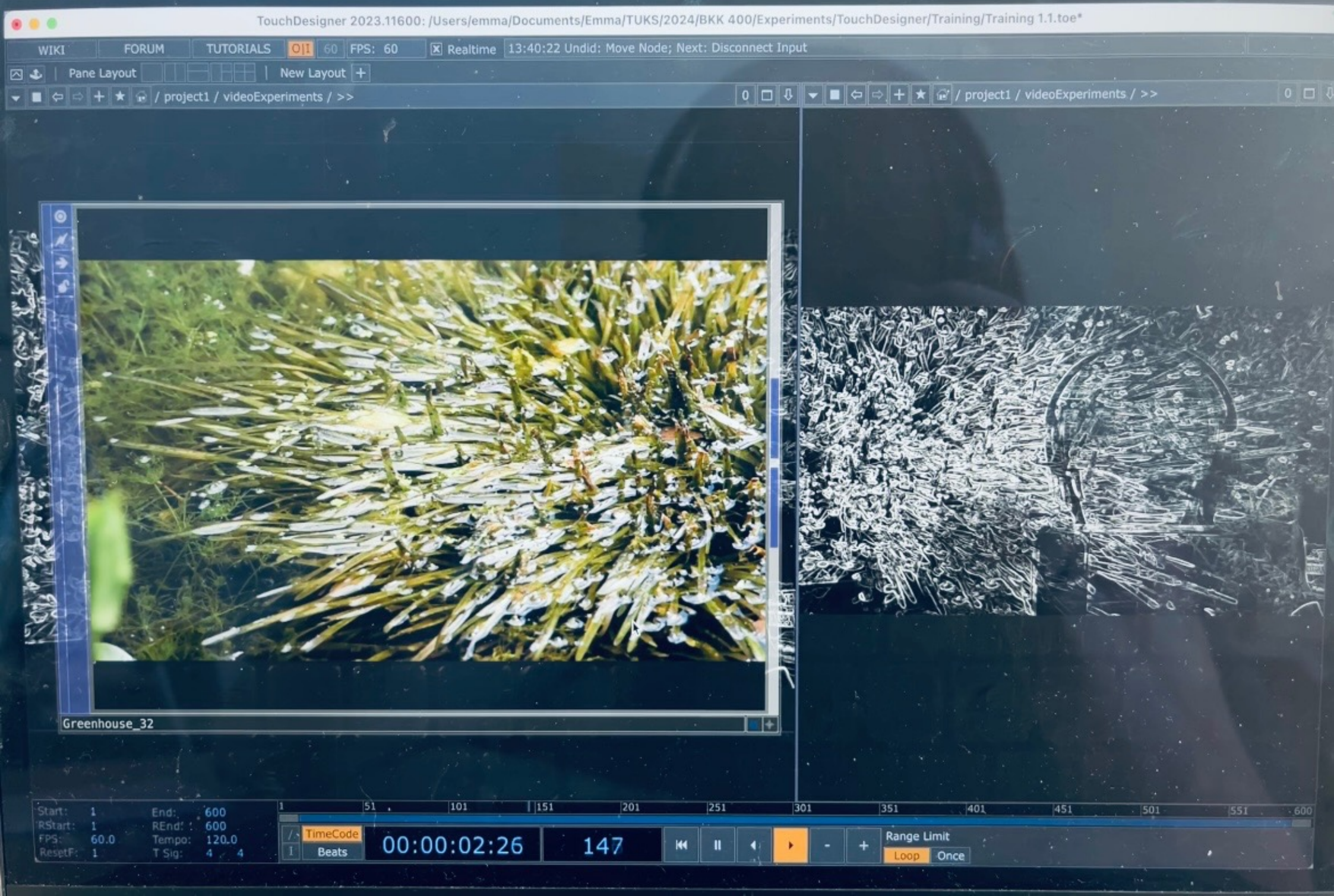
```
13 //--------------------------------------------------------------
14 void ofApp::draw(){ofDrawBitmapStringHighlight("Everything works!", 20, 20);
15
16     //Here is a good example of what took me a while to figure out: just because some of the functions are a little simpler
           doesn't mean they can't still produce interesting visual outcomes. This block of code uses quite straightforward functions
           and systems, I didn't even need to create a new class, but the animation is already more advanced than some of my previous
           demos. I've broken down each section to show what the line of code is controlling and basically once I learned what the
           different functions did here, I could compile them in original ways.
17
18     // Draw 100 circles
19     for(int i = 0; i < 100; i++) {
20         // Set animation speed
21         float speed = 2.0;
22
23         // set left-to-right oscillation
24         float sinX = sin(i + ofGetElapsedTimef() * speed) * 100;
25
26         // set opacity of each circle
27         float sinAlpha = ofMap(sin(i + ofGetElapsedTimef() * speed),-1, 1, 0, 200);
28
29         // set radius of each circle
30         float sinRadius = ofMap(sin(i * ofGetElapsedTimef() * 0.05),-1, 1, 5, 30);
31
32         // set color to white and opacity to sinAlpha
33         ofSetColor(255,255,255,sinAlpha);
34
35         // Finally draw each circle on window
36         ofDrawCircle(sinX + ofGetWidth()*0.5, i * 10, sinRadius);
37     }
38
39 }
40
41 //--------------------------------------------------------------
42 void ofApp::exit(){
43
44 }
45
46 //--------------------------------------------------------------
```

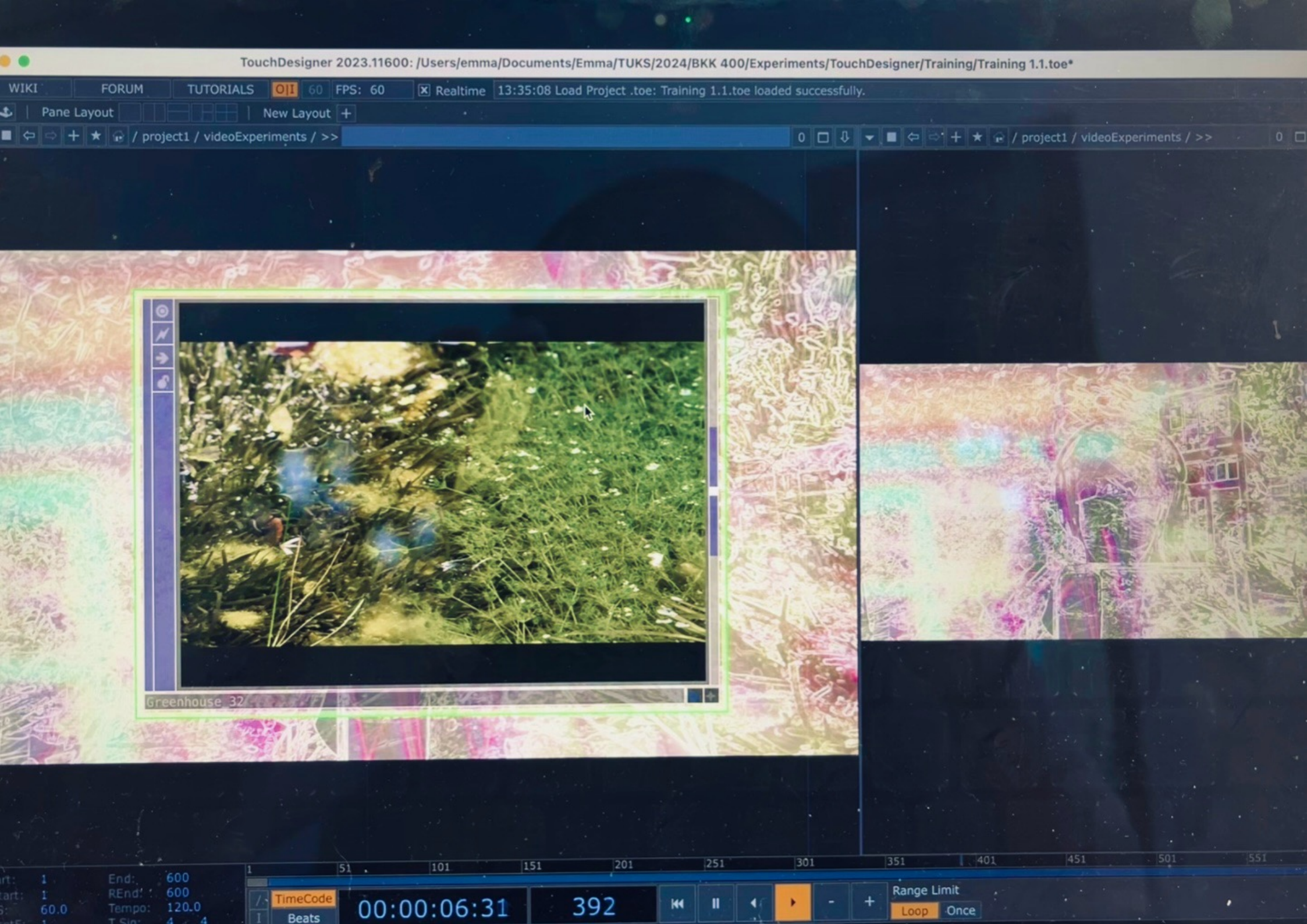


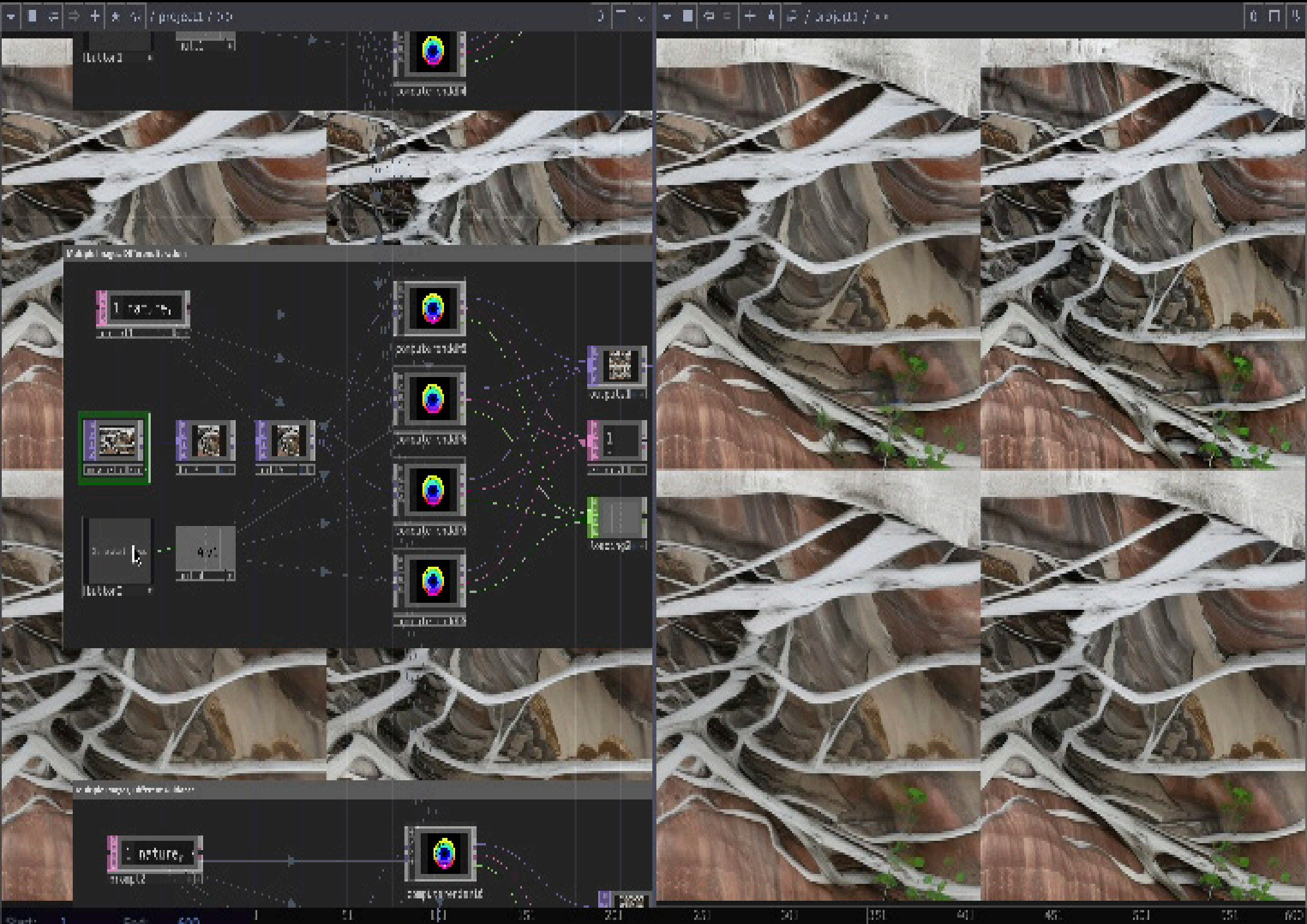
Touchdesigner

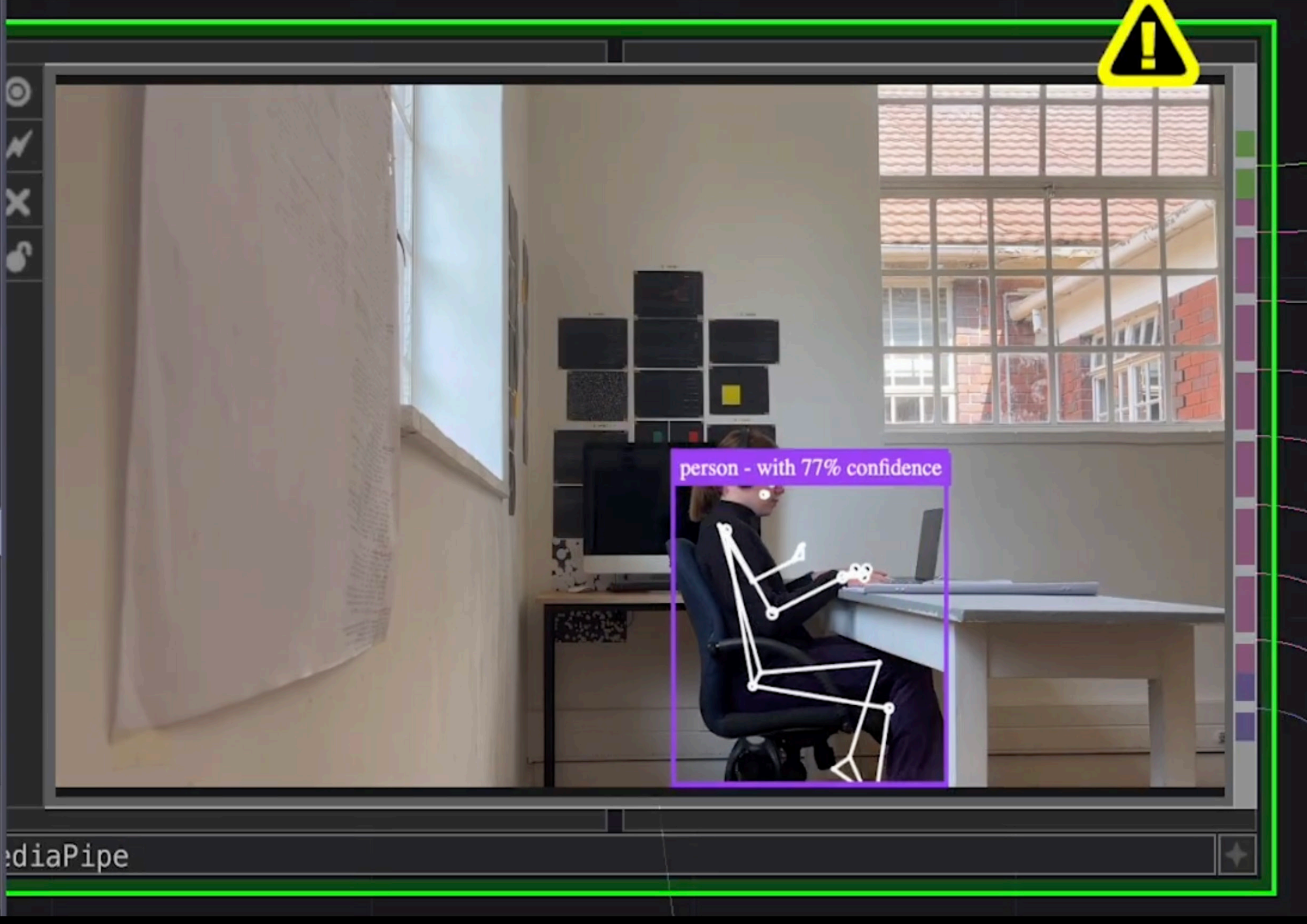

TouchDesigner 2023.11600: /Users/emma/Documents/Emma/TUKS/2024/BKK 400/Experiments/TouchDesigner/Training/Training 1.4.toe*
WIKI
FORUM
TUTORIALS
FPS: 40
Realtime
16:40:15 Load Project .toe: Training 1.4.toe loaded successfully.
Pane Layout
New Layout
/ project1 / particleSystem_1 / >>
render1
transform1
Line1
Start: 1
RStart: 1
FPS: 60.0
ResetF: 1
End: 600
REnd: 600
Tempo: 120.0
T Sig: 4 / 4
TimeCode
Beats
00:00:03:19
200
Range Limit
Loop
Once
MacBook Air


TouchDesigner 2023.11600: /Users/emma/Documents/Emma/TU65/2024/EKK 400/Experiments/TouchDesigner/Training/training 1.1.toe*
WIKI
FORUM
TUTORIALS
FPS: 60
Realtime
13:44:35 Undid: Connection; Next: Disconnect Input
Pane Layout
New Layout
/ project1 / videoExperiments / >>
/ project1 / videoExperiments / >>
comp2
Start: 1
End: 600
FPS: 60.0
Tempo: 120.0
T Sig: 4 4
00:00:07:23
444
Range Limit
Once
MacBook Air


TouchDesigner 2023.11600: /Users/emma/Documents/Emma/TUKS/2024/BKK 400/Experiments/TouchDesigner/Training/Training 1.1.toe*
WIKI
FORUM
TUTORIALS
FPS: 60
Realtime
13:40:22 Undid: Move Node; Next: Disconnect Input
Pane Layout
New Layout
/ project1 / videoExperiments / >>
/ project1 / videoExperiments / >>
Greenhouse_32
Start: 1
RStart: 1
FPS: 60.0
ResetF: 1
End: 600
REnd: 600
Tempo: 120.0
T Sig: 4 / 4
TimeCode
Beats
00:00:02:26
147
Range Limit
Loop
Once


TouchDesigner 2023.11600: /Users/emma/Documents/Emma/TUKS/2024/BKK 400/Experiments/TouchDesigner/Training/Training 1.1.toe*
WIKI
FORUM
TUTORIALS
FPS: 60
Realtime
13:35:08 Load Project .toe: Training 1.1.toe loaded successfully.
Pane Layout
New Layout
/ project1 / videoExperiments / >>
/ project1 / videoExperiments / >>
Greenhouse 32
End: 600
REnd: 600
Tempo: 120.0
TimeCode
Beats
00:00:06:31
392
Range Limit
Loop
Once




person - with 77% confidence
ediaPipe