Final project

Project description

The project used openframeworks for production. The knowledge points used are those learned in the first week to the fourth week of the course. The project is a comprehensive project on openframeworks. This project includes sound, image processing, 3d and other aspects.

Project presentation

The theme of this project is about "starry sky". The starry sky is a wonderful and mysterious place, with a certain magic. (In my work) I wanted to show a more magical way so I used a lot of image processing to make the image magic, including different 3d graphics moving all the time and so on.

The challenge of the project to me

This semester's learning compared to last semester's learning can feel the learning content has become more. For someone who hasn't studied code before, with the basics from last semester, working with openframeworks this semester feels much better than not knowing anything at all. Therefore, I have tried my best to complete this project. Although there is still a lot to be improved, I think continuous learning will lead to improvement. I will be ready to meet new challenges next semester!

Project reflection and thinking (Areas worth improving)

There are still many areas in need of improvement in this project. In the final inspection, it was found that a lot of codes were rather chaotic and needed some time to sort out and become more streamlined. In terms of effects, I think there's a lot more to explore, like thinking about building your own model.

In this project, I have made many attempts, and many code errors have made it impossible to proceed. I will think about these aspects in the future. For example, the effect of the 3d model unraveling that I wanted to show ended up being a little bit bad. The effect of these graphics spreading out feels a little too scattered, and I will consider starting from this aspect in the future when I have time. I need to learn more about how to use c++ and openframeworks better.

Code interpretation

```
void ofApp  setup
ofSetVerticalSync(true);

  eventString = "Alpha";
   vagRounded.load("Batang.ttf",25);

   blendMode = OF_BLENDMODE_ALPHA;
   ofDisableArbTex();

   texture.getTexture().setTextureWrap( GL_REPEAT, GL_REPEAT );
   vidGrabber.setup(640, 480, true);
   bFill       = true;
   bWireframe     true
   bDrawNormals   false
   bDrawAxes      false
   bDrawLights    false
   bHelpText      true
   bMousePressed      false
   bSplitFaces    false
   float width     = ofGetWidth() * .12;
   float height    = ofGetHeight() * .12;


   plane.set( width*0.7, height*0.7 );
   box.set( width*0.5 );
   sphere.setRadius( width*0.4 );
   icoSphere.setRadius( width*0.4 );
```

```
    cylinder.set(width*.3, height*0.8);
    cone.set( width*.35, height*0.9 );

    mode = 0;

    ofSetSmoothLighting(true);
    pointLight.setDiffuseColor( ofFloatColor( .85, .85, .55) );
    pointLight.setSpecularColor( ofFloatColor(1.f, 1.f, 1.f));

    pointLight2.setDiffuseColor( ofFloatColor( 238.f/255.f, 57.f/255.f, 135.f/255.f ));
    pointLight2.setSpecularColor(ofFloatColor(.8f, .8f, .9f));

    pointLight3.setDiffuseColor( ofFloatColor(19.f/255.f,94.f/255.f,77.f/255.f) );
    pointLight3.setSpecularColor( ofFloatColor(18.f/255.f,150.f/255.f,135.f/255.f) );


    material.setShininess( 120 );
    // the light highlight of the material //
    material.setSpecularColor(ofColor(255, 255, 255, 255));

    ofSetSphereResolution(24);



    ofBackground(4, 34, 34);
            int bufferSize          = 512;
            sampleRate                  44100
            phase                   = 0;
            phaseAdder                  0.4f
            phaseAdderTarget        0.0f
            volume                      0.1f
            bNoise                  = false;
    lAudio.assign(bufferSize, 0.9);
            rAudio.assign(bufferSize, 0.0);
            soundStream.printDeviceList();

            ofSoundStreamSettings settings;


#ifdef TARGET_LINUX

            auto devices = soundStream.getMatchingDevices("default");
            if(!devices.empty()){
                    settings.setOutDevice(devices[0]);
            }
#endif

            settings.setOutListener(this);
            settings.sampleRate = sampleRate;
            settings.numOutputChannels = 2;
            settings.numInputChannels = 0;
            settings.bufferSize = bufferSize;
            soundStream.setup(settings);



    blendMode = OF_BLENDMODE_ALPHA;



    int screenW = ofGetScreenWidth();
    int screenH = ofGetScreenHeight();

    ofBackground(0,0,0);
```

```
    // set up the boids
    for (int i = 0; i < 50; i++)
        boids.push_back(new Boid());

    for (int i = 0; i < 50; i++)
        myboids.push_back(new Boid());

    quality = OF_IMAGE_QUALITY_WORST;
    maxSize    1500
    glitchStart    0.1
    reset


}
```

ofSetVerticalSync(true) enables vertical synchronization, which synchronizes the application's frame rate with the vertical refresh rate of the display to prevent visual artifacts such as tearing.
eventString is initialized to the string "Alpha", and vagRounded is loaded with the font "Batang.ttf" and a size of 25.
blendMode is set to OF_BLENDMODE_ALPHA, which specifies that alpha blending should be used to composite colors.
ofDisableArbTex() disables the automatic texture coordinate normalization, which can improve performance for certain types of textures.
texture is loaded and its texture wrapping is set to repeat in both directions using GL_REPEAT.
vidGrabber is set up to grab frames from a video input source with a resolution of 640x480 and using the default video device.
Several shapes are created using ofBoxPrimitive, ofPlanePrimitive, ofSpherePrimitive, ofIcoSpherePrimitive, ofCylinderPrimitive, and ofConePrimitive, and their sizes are set based on the application window size.
Three ofLight objects and an ofMaterial object are created and initialized with various colors and properties.
The smooth lighting option is enabled with ofSetSmoothLighting(true).
An ofSoundStream object is set up to handle audio output, with settings for sample rate, buffer size, and number of output channels.
The Boid class is used to create two vectors of boids, with 50 Boid objects in each vector.
Several variables related to glitch effects are initialized with default values.
Finally, the background color is set to a dark greenish-blue color using ofBackground().

```
void ofApp  reset
    generation    0

    // pick random for bleeding
    addX = ofRandom(20, 20);
    addY = ofRandom(10, 60);
    subX = ofRandom(15, addX);
    subY = ofRandom(0, addY-20);
}
```

generation is reset to 0.
addX is assigned a random number between 20 and 20.
addY is assigned a random number between 10 and 60.
subX is assigned a random number between 15 and addX.
subY is assigned a random number between 0 and ady-20.

```
void ofApp  update
    pointLight.setPosition((ofGetWidth()*.5)+ cos(ofGetElapsedTimef()*.5)*(ofGetWidth()*.3), ofGetHeight()/2, 500);
    pointLight2.setPosition((ofGetWidth()*.5)+ cos(ofGetElapsedTimef()*.15)*(ofGetWidth()*.3),
                            ofGetHeight()*.5 + sin(ofGetElapsedTimef()*.7)*(ofGetHeight()), -300);

    pointLight3.setPosition(
                            cos(ofGetElapsedTimef()*1.5) * ofGetWidth()*.5,
                            sin(ofGetElapsedTimef()*1.5f) * ofGetWidth()*.5,
                            cos(ofGetElapsedTimef()*.2) * ofGetWidth()
                            );

    //ofSetWindowTitle("Framerate: "+ofToString(ofGetFrameRate(), 0));
    if(mode == 2 || ofGetElapsedTimef() < 10) {
        vidGrabber.update();

    }




    ofVec3f min(0, 0);
    ofVec3f max(ofGetWidth(), ofGetHeight());
    for (int i = 0; i < boids.size(); i++)
    {
        boids[i]->update(boids, min, max);
    }
    for (int i = 0; i < myboids.size(); i++)
    {
        myboids[i]->update(myboids, min, max);
    }



    string curFilename = "xingkong2.jpg";

    int size = img.getWidth();

    // keeps the image from getting too big
    if(size < maxSize) {
        img.save(curFilename, quality);

        if(ofGetKeyPressed('g')) {
            // this portion glitches the jpeg file
            // first loading the file
            ofBuffer file = ofBufferFromFile(curFilename, true);
            int fileSize = file.size();
            char * buffer = file.getData();

            // pick a byte offset that is somewhere near the end of the file
            int whichByte = (int) ofRandom(fileSize * glitchStart, fileSize);
            // and pick a bit in that byte to turn on
            int whichBit = ofRandom(8);
            char bitMask = 1 << whichBit;
            // using the OR operator
            buffer[whichByte] |= bitMask;

            // write the file out like nothing happened
            ofBufferToFile(curFilename, file, true);
            img.load(curFilename);
        } else {
            img.load(curFilename);

            // switches every other frame
            // resizing up and down breaks the 8x8 JPEG blocks
```

```
            if(ofGetFrameNum() % 2 == 0) {
                // resize a little bigger
                img.resize(size + addX, size + addY);
            } else {
                // then resize a little smaller
                img.resize(size - subX, size - subY);
            }
        }
        generation
    }
}
```

pointLight and pointLight2 objects are being updated to create the effect of a moving light source. pointLight moves horizontally based on the cosine function, while pointLight2 moves horizontally and vertically based on the cosine and sine functions respectively. pointLight3 moves in a circular pattern using the cosine and sine functions as well.
If mode equals 2 or the elapsed time is less than 10, then the vidGrabber object is being updated. Two for loops are used to update objects in the boids and myboids vectors using their update() methods.
An image is being saved to a file named "xingkong2.jpg". If the size of the image is less than maxSize, then the image is saved with a given quality setting. If the 'g' key is pressed, the image file is glitched, where a random byte offset is chosen and a bit in that byte is turned on. If the 'g' key is not pressed, then the image is loaded and every other frame, the image is resized to be a little larger or smaller based on the addX, addY, subX, and subY variables. Finally, the generation variable is incremented.

```
void ofApp  draw


    // draw the original image
    ofSetColor(ofColor::white);
    img.draw(0, 0);

    // draw the four rectangles
    ofNoFill();
    ofSetColor(ofColor::red);
    ofDrawCircle(mouseX, mouseY, 30, 30);

    ofSetColor(ofColor::green);
    ofDrawRectangle(mouseX, mouseY, 40, 30);

    ofSetColor(ofColor::blue);
    ofDrawRectangle(mouseX + 25, mouseY + 25, 20, 20);

    ofSetColor(ofColor::magenta);
    ofDrawRectangle(mouseX-7, mouseY-7 , 20, 20);

    // draw the four corresponding subsections

    ofSetColor(ofColor::white);
    img.drawSubsection(0, 0, 100, 100, mouseX, mouseY);
    ofSetColor(ofColor::red);
    ofDrawRectangle(0, 0, 100, 100);

    ofSetColor(ofColor::white);
    img.drawSubsection(0, 100, 100, 100, mouseX, mouseY, 50, 50);
    ofSetColor(ofColor::green);
    ofDrawRectangle(0, 100, 100, 100);
```

```
    ofSetColor(ofColor::white);
    img.drawSubsection(0, 200, 100, 100, mouseX + 25, mouseY + 25, 50, 50);
    ofSetColor(ofColor::blue);
    ofDrawRectangle(0, 200, 100, 100);

    ofSetColor(ofColor::white);
    img.drawSubsection(0, 300, 100, 100, mouseX - 7, mouseY - 7, 50, 50);
    ofSetColor(ofColor::magenta);
    ofDrawRectangle(0, 300, 100, 100);
```

It is responsible for drawing the original image and four rectangles, as well as four corresponding subsections of the image.
First, the original image is drawn at the top left corner of the window with img.draw(0, 0).
Next, four colored rectangles are drawn using ofDrawRectangle at the current mouse position and other positions based on the mouse position. The rectangles are colored red, green, blue, and magenta, respectively.
Finally, four corresponding subsections of the image are drawn using img.drawSubsection, with the color of each subsection corresponding to the color of its corresponding rectangle. The subsections are drawn at specific positions relative to the rectangles and are sized differently. Each subsection is preceded by a white rectangle drawn using ofDrawRectangle.

```
ofEnableBlendMode(OF_BLENDMODE_ALPHA);
    ofSetColor(265);
                        "(s): Unpause the audio\n(e): Pause the audio\n(f): Toggle Fullscreen\n(o): Draw Solid
Shapes\n(w): Draw Wireframes\n(1/2/3/4): Set Resolutions\n(n): Draw Normals\n(LEFT/RIGHT): Set Mode" 30 40
                        "(z): Split Faces\n(a): Draw Axes\n(l): Render lights\n(h): Toggle help\n(g): Glitch" 400 40
    ofEnableBlendMode(OF_BLENDMODE_ALPHA);
    ofSetHexColor(0xffffff);
    vagRounded.drawString(eventString, 98,198);
    ofSetHexColor(0xffffff);
    vagRounded.drawString(eventString, 98,198);
    ofSetColor(255,122,220);
    vagRounded.drawString(eventString, 100,200);
    ofSetHexColor(0xffffff);
    vagRounded.drawString(timeString, 98,98);
    ofSetColor(255,122,220);
    vagRounded.drawString(timeString, 100,100);
    ofSetColor(255, 255, 255,255);


    ofEnableBlendMode(blendMode);



    ofDisableBlendMode();
```

This code sets the blend mode to alpha, which allows transparency and color overlap.
Set the color to white and the alpha value to 265.
Draws two strings of text on the screen, one on the left and one on the right, showing the various keyboard controls and Settings for the application.
Using the first string called vagRounded. The eventString is drawn twice and slightly offset to create the drop shadow effect, and then again with a different color to create the highlighted effect.
The second string, timeString, is also drawn twice with the shadow effect and then a different color.
Set the blend mode to the value of the variable blendMode.

```
ofNoFill();

        // draw the left channel:
        ofPushStyle();
                ofPushMatrix();
                ofTranslate(60, 150, 0);

                ofSetColor(265);
                ofDrawBitmapString("Left Channel", 400, 18);

                ofSetLineWidth(0.2);
                ofDrawRectangle(0, 0, 900, 200);

                ofSetColor(25, 218, 225);
                ofSetLineWidth(3);

                        ofBeginShape();
                        for (unsigned int i = 0; i < lAudio.size(); i++){
                                float x =  ofMap(i, 0, lAudio.size(), 0, 900, true);
                                ofVertex(x, 100 -lAudio[i]*180.0f);
                        }
                        ofEndShape(false);

                ofPopMatrix();
        ofPopStyle();

        // draw the right channel:
        ofPushStyle();
                ofPushMatrix();
                ofTranslate(60, 350, 0);

                ofSetColor(265);
                ofDrawBitmapString("Right Channel", 400, 18);

                ofSetLineWidth(1);
                ofDrawRectangle(0, 0, 900, 200);

                ofSetColor(245, 18, 225);
                ofSetLineWidth(3);

                        ofBeginShape();
                        for (unsigned int i = 0; i < rAudio.size(); i++){
                                float x =  ofMap(i, 0, rAudio.size(), 0, 900, true);
                                ofVertex(x, 100 -rAudio[i]*180.0f);
                        }
                        ofEndShape(false);

                ofPopMatrix();
        ofPopStyle();


        ofSetColor(265);
                        " (-/+) : ("                    2   ") modify volume\n (x) : ("                    2   ")
modify pan\n (y) : "
        if   bNoise
                reportString += "modify sine wave (" + ofToString(targetFrequency, 2) + "hz)";
         else
                reportString += "noise";
        }
        ofDrawBitmapString(reportString, 32, 579);
```

ofNoFill(); sets the drawing mode to outline mode, so that shapes drawn later will not be filled. The code then draws two plots, one for the left audio channel and one for the right audio channel. Each plot consists of:

A label indicating the channel name, drawn with ofDrawBitmapString.
A rectangle representing the plot area, drawn with ofDrawRectangle.
A waveform plot drawn with ofBeginShape, ofVertex, and ofEndShape. The waveform is generated by iterating through the audio data in the channel (lAudio or rAudio), mapping the sample index to the x-coordinate of the plot, and mapping the sample value to the y-coordinate of the plot.
Finally, a report string is drawn at the bottom of the screen, indicating the current volume, pan, and waveform type (either sine wave or noise). The report string is generated using ofDrawBitmapString and ofToString.

```
float spinX = sin(ofGetElapsedTimef()*.35f);
    float spinY = cos(ofGetElapsedTimef()*.075f);

    if  bMousePressed
        spinX = spinY = 0.0f;
    }

    cam.setGlobalPosition({ 0,0,cam.getImagePlaneDistance(ofGetCurrentViewport()) });
    cam.begin();

    ofEnableDepthTest();

    ofEnableLighting();
    pointLight.enable();
    pointLight2.enable();
    pointLight3.enable();




    if (mode == 1 || mode == 3) texture.getTexture().bind();
    if (mode == 2) vidGrabber.getTexture().bind();

    float screenWidth = ofGetWidth();
    float screenHeight = ofGetHeight();

    plane.setPosition(     -screenWidth * .5 + screenWidth *  0.8/9.f, screenHeight *  -2/5.f, 0);
    box.setPosition(       -screenWidth * .5 + screenWidth *  1.7/3.f, screenHeight *  -2/5.f, 0);
    sphere.setPosition(    -screenWidth * .5 + screenWidth *  2.7/3.f, screenHeight *  -2/5.f, 0);
    icoSphere.setPosition( -screenWidth * .5 + screenWidth *  1/4.f, screenHeight * -2/5.f, 0);
    cylinder.setPosition(  -screenWidth * .5 + screenWidth *  1.2/3.f, screenHeight * -2/5.f, 0);
    cone.setPosition(      -screenWidth * .5 + screenWidth *  2.9/4.f, screenHeight * -2/5.f, 0);

    // Plane //

    plane.rotateDeg(spinX, 1.0, 0.0, 0.0);
    plane.rotateDeg(spinY, 0, 1.0, 0.0);



    if (mode == 3) {
        deformPlane = plane.getMesh();
        // x = columns, y = rows //
        glm::vec2 planeDims = plane.getResolution();
        float planeAngleX = ofGetElapsedTimef()*3.6;
        float planeAngleInc = 3.f / (float)planeDims.x;
        glm::vec3 vert;
        for (size_t i = 0; i < deformPlane.getNumIndices(); i++) {
            planeAngleX += planeAngleInc;
            int ii = deformPlane.getIndex(i);
            vert = deformPlane.getVertex(ii);
            vert.z += cos(planeAngleX) * 50;
```

```
        deformPlane.setVertex(ii, vert);
    }
}

if   bFill       bWireframe

    material.begin();
}

if  bFill
    material.begin();
    ofFill();
    if (mode == 3) {
        plane.transformGL();
        deformPlane.draw();
        plane.restoreTransformGL();
    }
    else {
        plane.draw();
    }
    material.end();
}

if  bWireframe
    ofNoFill();
    ofSetColor(0, 0, 0);
    plane.setPosition(plane.getPosition().x, plane.getPosition().y, plane.getPosition().z + 1);
    plane.drawWireframe();
    plane.setPosition(plane.getPosition().x, plane.getPosition().y, plane.getPosition().z - 1);

}

// Box //

box.rotateDeg(spinX, 1.0, 0.0, 0.0);
box.rotateDeg(spinY, 0, 1.0, 0.0);

if  bFill
    material.begin();
    ofFill();
    if (mode == 3) {
        box.transformGL();
        for (int i = 0; i < ofBoxPrimitive::SIDES_TOTAL; i++) {
            ofPushMatrix();
            ofTranslate(boxSides[i].getNormal(0) * sin(ofGetElapsedTimef()) * 50);
            boxSides[i].draw();
            ofPopMatrix();
        }
        box.restoreTransformGL();
    }
    else {
        box.draw();
    }
    material.end();
}

if  bWireframe
    ofNoFill();
    ofSetColor(0, 0, 0);
    box.setScale(1.01f);
    box.drawWireframe();
    box.setScale(1.f);
}
```

It's a 3D scene that can render different original shapes that are located in 3D space and rotated according to time-dependent sine and cosine functions.

The camera is positioned at a fixed distance from the image plane and is set to enable depth testing, lighting, and 3-point light.

The code also includes the option to fill the shape with material and/or display its wireframes. If mode is set to 3, the shapes are deformed using mesh transformation techniques and their vertices are adjusted according to the sine function.

```
// Sphere //
    sphere.rotateDeg(spinX, 1.0, 0.0, 0.0);
    sphere.rotateDeg(spinY, 0, 1.0, 0.0);

    if (mode == 3) {
        sphere.setMode(OF_PRIMITIVE_TRIANGLES);
        triangles = sphere.getMesh().getUniqueFaces();
    }

    if  bFill
        material.begin();
        ofFill();
        if (mode == 3) {
            float angle = ofGetElapsedTimef()*3.2;
            float strength = (sin(angle + .25)) * .5f * 5.f;
            glm::vec3 faceNormal;
            for (size_t i = 0; i < triangles.size(); i++) {

                faceNormal = triangles[i].getFaceNormal();
                for (int j = 0; j < 3; j++) {
                    triangles[i].setVertex(j, triangles[i].getVertex(j) + faceNormal * strength);
                }
            }
            sphere.getMesh().setFromTriangles(triangles);
        }
        sphere.draw();
        material.end();
    }

    if  bWireframe
        ofNoFill();
        ofSetColor(0, 0, 0);
        sphere.setScale(1.01f);
        sphere.drawWireframe();
        sphere.setScale(1.f);
    }


    // ICO Sphere //

    icoSphere.rotateDeg(spinX, 1.0, 0.0, 0.0);
    icoSphere.rotateDeg(spinY, 0, 1.0, 0.0);

    if (mode == 3) {
        triangles = icoSphere.getMesh().getUniqueFaces();
    }

    if  bFill
        material.begin();
        ofFill();

        if (mode == 3) {
            float angle = (ofGetElapsedTimef() * 1.4);
```

```
            glm::vec3 faceNormal;
            for (size_t i = 0; i < triangles.size(); i++) {
                float frc = ofSignedNoise(angle* (float)i * .1, angle*.05) * 4;
                faceNormal = triangles[i].getFaceNormal();
                for (int j = 0; j < 3; j++) {
                    triangles[i].setVertex(j, triangles[i].getVertex(j) + faceNormal * frc);
                }
            }
            icoSphere.getMesh().setFromTriangles(triangles);
        }

        icoSphere.draw();
        material.end();
    }

    if  bWireframe
        ofNoFill();
        ofSetColor(0, 0, 0);
        icoSphere.setScale(1.01f);
        icoSphere.drawWireframe();
        icoSphere.setScale(1.f);
    }


    // Cylinder //
    if (mode == 3) {
        topCap = cylinder.getTopCapMesh();
        bottomCap = cylinder.getBottomCapMesh();
        body = cylinder.getCylinderMesh();
    }

    cylinder.rotateDeg(spinX, 1.0, 0.0, 0.0);
    cylinder.rotateDeg(spinY, 0, 1.0, 0.0);
    if  bFill
        material.begin();
        ofFill();
        if (mode == 3) {
            cylinder.transformGL();
            ofPushMatrix(); {
                if (topCap.getNumNormals() > 0) {
                    ofTranslate(topCap.getNormal(0) * (cos(ofGetElapsedTimef() * 5) + 1)*.5f * 100);
                    topCap.draw();
                }
            } ofPopMatrix();
            ofPushMatrix(); {
                if (bottomCap.getNumNormals() > 0) {
                    ofTranslate(bottomCap.getNormal(0) * (cos(ofGetElapsedTimef() * 4) + 1)*.5f * 100);
                    bottomCap.draw();
                }
            } ofPopMatrix();
            ofPushMatrix(); {
                float scale = (cos(ofGetElapsedTimef() * 3) + 1)*.5f + .2;
                ofScale(scale, scale, scale);
                body.draw();
            } ofPopMatrix();
            cylinder.restoreTransformGL();
        }
        else {
            cylinder.draw();
        }
        material.end();
    }

    if  bWireframe
        ofNoFill();
        ofSetColor(0, 0, 0);
```

```
    cylinder.setScale(1.01f);
    cylinder.drawWireframe();
    cylinder.setScale(1.0f);
}


// Cone //
cone.rotateDeg(spinX, 1.0, 0.0, 0.0);
cone.rotateDeg(spinY, 0, 1.0, 0.0);

if (mode == 3) {
    bottomCap = cone.getCapMesh();
    body = cone.getConeMesh();
}
if  bFill
    material.begin();
    ofFill();
    if (mode == 3) {
        cone.transformGL();
        ofPushMatrix();
        if (bottomCap.getNumNormals() > 0) {
            ofTranslate(bottomCap.getNormal(0) * cone.getHeight()*.5);
            ofRotateDeg(sin(ofGetElapsedTimef() * 5) * RAD_TO_DEG, 1, 0, 0);
            bottomCap.draw();
        }
        ofPopMatrix();

        ofPushMatrix();
        ofRotateDeg(90, 1, 0, 0);
        ofRotateDeg((cos(ofGetElapsedTimef() * 6) + 1)*.5 * 360, 1, 0, 0);
        body.draw();
        ofPopMatrix();
        cone.restoreTransformGL();
    }
    else {
        cone.draw();
    }
    material.end();
}

if  bWireframe
    ofNoFill();
    ofSetColor(0, 0, 0);
    cone.setScale(1.01f);
    cone.drawWireframe();
    cone.setScale(1.0f);
}

if   bFill       bWireframe
    material.end();
}

if (mode == 1 || mode == 3) texture.getTexture().unbind();
if (mode == 2) vidGrabber.getTexture().unbind();

material.end();
ofDisableLighting();

if  bDrawLights
    ofFill();
    ofSetColor(pointLight.getDiffuseColor());
    pointLight.draw();
    ofSetColor(pointLight2.getDiffuseColor());
    pointLight2.draw();
    ofSetColor(pointLight3.getDiffuseColor());
    pointLight3.draw();
```

```
    }

    if  bDrawNormals
        ofSetColor(225, 0, 255);
        plane.drawNormals(20, bSplitFaces);
        box.drawNormals(20, bSplitFaces);
        sphere.drawNormals(20, bSplitFaces);
        icoSphere.drawNormals(20, bSplitFaces);
        cylinder.drawNormals(20, bSplitFaces);
        cone.drawNormals(20, bSplitFaces);
    }
    if  bDrawAxes
        plane.drawAxes(plane.getWidth()*.5 + 30);
        box.drawAxes(box.getWidth() + 30);
        sphere.drawAxes(sphere.getRadius() + 30);
        icoSphere.drawAxes(icoSphere.getRadius() + 30);
        cylinder.drawAxes(cylinder.getHeight() + 30);
        cone.drawAxes(cone.getHeight() + 30);
    }

    ofDisableDepthTest();

    ofFill();

    cam.end();

    ofSetDrawBitmapMode(OF_BITMAPMODE_MODEL_BILLBOARD);
```

The code defines several shapes -- a sphere, an icosahedral sphere, a cylinder, and a cone. The code first rotates the sphere and icosahedral sphere spinY according to the values of spinX and. If the mode variable is set to 3, the sphere object is set to use the triangular primitive and retrieve the unique face of its grid. Similarly, if the mode variable is set to 3, icoSphere retrieves the unique face of the object grid.
If the bFill variable is true, the material is started and the shape is filled with the selected color. If the mode variable is set to 3, the faces of the sphere or icosahedral sphere are deformed using the noise function and the resulting grid is displayed. If the bWireframe variable is true, the shape appears as a wireframe.
The cylinder rotates according to spinX and, if set to 3, spinY retrieves its top cover, bottom cover, and cylinder grid. If true, the cylinder will be filled with the selected material. If set to 3, the noise function is used to deform the top and bottom covers of the cylinder and display the resulting grid.
modebFillmode
Finally, the cone rotates according to spinX and spinY. If mode is set to 3, the bottom cover of the cone and the cone grid are retrieved. If bFill is true, the cone will fill the selected material. If mode is set to 3, the noise function is used to deform the bottom cover of the cone and display the resulting grid.

```
void ofApp  keyPressed   int
    switch (key) {
        case 49:
            blendMode = OF_BLENDMODE_ALPHA;
            eventString = "Alpha";
            break;
        case 50:
            blendMode = OF_BLENDMODE_ADD;
            eventString = "Add";
            break;
        case 51:
            blendMode = OF_BLENDMODE_MULTIPLY;
            eventString = "Multiply";
```

```
        break;
    case 52:
        blendMode = OF_BLENDMODE_SUBTRACT;
        eventString = "Subtract";
        break;
    case 53:
        blendMode = OF_BLENDMODE_SCREEN;
        eventString = "Screen";
        break;
    default:
        break;
    case 'f':
        ofToggleFullscreen();
        break;
    case 'o':
        bFill = !bFill;
        break;
    case 'w':
        bWireframe = !bWireframe;
        break;
    case '49':
        bSplitFaces=false;
        sphere.setResolution(4);
        // icosahedron //
        icoSphere.setResolution(0); // number of subdivides //
        plane.setResolution(3, 2);
        cylinder.setResolution(4,2,0);
        cone.setResolution(4, 1, 0);
        box.setResolution(1);
        break;
    case '50':
        bSplitFaces=false;
        sphere.setResolution(8);
        icoSphere.setResolution(1);
        plane.setResolution(6, 4);
        cylinder.setResolution(8,4,1);
        cone.setResolution(7, 2, 1);
        box.setResolution(2);
        break;
    case '51':
        bSplitFaces=false;
        sphere.setResolution(16);
        icoSphere.setResolution(2);
        plane.setResolution(8,5);
        cylinder.setResolution(12, 9, 2);
        cone.setResolution(10, 5, 2);
        box.setResolution(6);
        break;
    case '52':
        bSplitFaces=false;
        sphere.setResolution(48);
        icoSphere.setResolution(4);
        plane.setResolution(12, 9);
        cylinder.setResolution(20, 13, 4);
        cone.setResolution(20, 9, 3);
        box.setResolution(10);
        break;
    case 'n':
        bDrawNormals = !bDrawNormals;
        break;
    case OF_KEY_RIGHT:
        mode++;
        if(mode > 3) mode = 0;
        if(mode==3){
            // to get unique triangles, you have to use triangles mode //
            sphere.setMode( OF_PRIMITIVE_TRIANGLES );
```

```
        }
        break;
    case OF_KEY_LEFT:
        mode--;
        if(mode < 0) mode = 3;
        if(mode==3){
            // to get unique triangles, you have to use triangles mode //
            sphere.setMode( OF_PRIMITIVE_TRIANGLES );
        }
        break;
    case 'a':
        bDrawAxes = !bDrawAxes;
        break;
    case 'l':
        bDrawLights = !bDrawLights;
        break;
    case 'h':
        bHelpText=!bHelpText;
        break;
    case 'z':
        bSplitFaces = !bSplitFaces;
        break;
}
        if (key == '-' || key == '_' ){
                volume      0.05
                volume = MAX(volume, 0);
        } else if (key == '+' || key == '=' ){
                volume      0.05
                volume = MIN(volume, 1);
        }

        if( key == 's' ){
                soundStream.start();
        }

        if( key == 'e' ){
                soundStream.stop();




    if mode      3  bSplitFaces    false

    if bSplitFaces
        sphere.setMode( OF_PRIMITIVE_TRIANGLES );
        vector<ofMeshFace> triangles = sphere.getMesh().getUniqueFaces();
        sphere.getMesh().setFromTriangles( triangles, true );

        icoSphere.setMode( OF_PRIMITIVE_TRIANGLES );
        triangles = icoSphere.getMesh().getUniqueFaces();
        icoSphere.getMesh().setFromTriangles(triangles, true);

        plane.setMode( OF_PRIMITIVE_TRIANGLES );
        triangles = plane.getMesh().getUniqueFaces();
        plane.getMesh().setFromTriangles(triangles, true);

        cylinder.setMode( OF_PRIMITIVE_TRIANGLES );
        triangles = cylinder.getMesh().getUniqueFaces();
        cylinder.getMesh().setFromTriangles(triangles, true);

        cone.setMode( OF_PRIMITIVE_TRIANGLES );
        triangles = cone.getMesh().getUniqueFaces();
        cone.getMesh().setFromTriangles(triangles, true);

        box.setMode( OF_PRIMITIVE_TRIANGLES );
```

```
            triangles = box.getMesh().getUniqueFaces();
            box.getMesh().setFromTriangles(triangles, true);

        } else {
            // vertex normals are calculated with creation, set resolution //
            sphere.setResolution( sphere.getResolution() );

            icoSphere.setResolution( icoSphere.getResolution() );
            plane.setResolution( plane.getNumColumns(), plane.getNumRows() );

            cylinder.setResolution( cylinder.getResolutionRadius(), cylinder.getResolutionHeight(),
cylinder.getResolutionCap() );
            cone.setResolution( cone.getResolutionRadius(), cone.getResolutionHeight(), cone.getResolutionCap() );
            box.setResolution( box.getResolutionWidth() );
        }

}

if mode      1
    // resize the plane to the size of the texture //
    plane.resizeToTexture( texture.getTexture() );
    // setTexCoordsFromTexture sets normalized or non-normalized tex coords based on an ofTexture passed in.
    box.mapTexCoordsFromTexture( texture.getTexture() );
    sphere.mapTexCoordsFromTexture( texture.getTexture() );
    icoSphere.mapTexCoordsFromTexture( texture.getTexture() );
    cylinder.mapTexCoordsFromTexture( texture.getTexture() );
    cone.mapTexCoordsFromTexture( texture.getTexture() );
}

if mode      2
    plane.resizeToTexture( vidGrabber.getTexture(), .5 );
    box.mapTexCoordsFromTexture( vidGrabber.getTexture() );
    sphere.mapTexCoordsFromTexture( vidGrabber.getTexture() );
    icoSphere.mapTexCoordsFromTexture( vidGrabber.getTexture() );
    cylinder.mapTexCoordsFromTexture( vidGrabber.getTexture() );
    cone.mapTexCoordsFromTexture( vidGrabber.getTexture() );
}

//
if( mode == 3 ) {

    bSplitFaces    false

    /
    sphere.setMode(OF_PRIMITIVE_TRIANGLE_STRIP);
    icoSphere.setMode(OF_PRIMITIVE_TRIANGLE_STRIP);
    cylinder.setMode( OF_PRIMITIVE_TRIANGLE_STRIP );
    cone.setMode( OF_PRIMITIVE_TRIANGLE_STRIP );

    box.setMode( OF_PRIMITIVE_TRIANGLES );

    plane.setMode( OF_PRIMITIVE_TRIANGLE_STRIP );
    plane.mapTexCoords(0, 0, 5, 5);

    // rebuild the box,
    box.mapTexCoords(0, 0, 5, 5);
    sphere.mapTexCoords(0, 0, 5, 5);
    icoSphere.mapTexCoords(0, 0, 5, 5);
    cylinder.mapTexCoords(0, 0, 5, 5);
    cone.mapTexCoords(0, 0, 5, 5);

    // store the box sides  //
    for(int i = 0; i < ofBoxPrimitive::SIDES_TOTAL; i++ ) {
        boxSides[i] = box.getSideMesh( i );
```

```
    }



        }

}
```

The keyPressed function is called whenever a key is pressed. It checks which key was pressed using a switch statement and performs different actions depending on the key.
If the key is a number between 1 and 5, it changes the blending mode used to draw the shapes on the screen. If it is the 'f' key, it toggles fullscreen mode. If it is the 'o' key, it toggles whether shapes are filled or wireframe. If it is the 'w' key, it toggles whether shapes are drawn as wireframes. If it is the 'n' key, it toggles whether normals are drawn. If it is the 'a' key, it toggles whether axes are drawn. If it is the 'l' key, it toggles whether lights are drawn. If it is the 'h' key, it toggles whether help text is displayed. If it is the 'z' key, it toggles whether faces are split.
If the key is the '-' or '_' key, it decreases the volume of a sound being played. If it is the '+' or '=' key, it increases the volume of a sound being played. If it is the 's' key, it starts playing the sound. If it is the 'e' key, it stops playing the sound and modifies the mode of the sphere and its subdivisions. If the mode is 1, it resizes the plane to the size of a texture and sets its texture coordinates from the texture.

```
void ofApp  keyReleased   int

}

//--------------------------------------------------------------
void ofApp  mouseMoved int    int
        int width = ofGetWidth();
        pan = (float)x / (float)width;
        float height = (float)ofGetHeight();
        float heightPct = ((height-y) / height);
        targetFrequency    2000.0f
        phaseAdderTarget = (targetFrequency / (float) sampleRate) * TWO_PI;
}

//--------------------------------------------------------------
void ofApp  mouseDragged int    int    int
        int width = ofGetWidth();
        pan = (float)x / (float)width;
}

//--------------------------------------------------------------
void ofApp  mousePressed int    int    int
        bNoise    true
            "xingkong.jpg"
}


//--------------------------------------------------------------
void ofApp  mouseReleased int    int    int
        bNoise    false
            "xingkong4.jpeg"
}

//--------------------------------------------------------------
void ofApp  mouseEntered int    int

}
```

```
//--------------------------------------------------------------
void ofApp  mouseExited int    int

}

//--------------------------------------------------------------
void ofApp  windowResized int    int

}

//--------------------------------------------------------------
void ofApp::audioOut(ofSoundBuffer & buffer){
        //pan = 0.5f;
        float leftScale = 1 - pan;
        float rightScale = pan;


        while (phase > TWO_PI){
                phase -= TWO_PI;
        }

        if  bNoise     true
                // ---------------------- noise --------------
                for (size_t i = 0; i < buffer.getNumFrames(); i++){
                        lAudio[i] = buffer[i*buffer.getNumChannels()    ] = ofRandom(0, 1) * volume * leftScale;
                        rAudio[i] = buffer[i*buffer.getNumChannels() + 1] = ofRandom(0, 1) * volume * rightScale;
                }
          else
                phaseAdder    0.95f   phaseAdder    0.05f   phaseAdderTarget
                for (size_t i = 0; i < buffer.getNumFrames(); i++){
                        phase += phaseAdder;
                        float sample = sin(phase);
                        lAudio[i] = buffer[i*buffer.getNumChannels()    ] = sample * volume * leftScale;
                        rAudio[i] = buffer[i*buffer.getNumChannels() + 1] = sample * volume * rightScale;



                }
        }

}
```

The mouseMoved function is called whenever the mouse is moved within the application window. It calculates the horizontal position of the mouse within the window (normalized between 0 and 1) and uses that to set the pan variable, which controls the balance between the left and right channels of the audio output. It also calculates the vertical position of the mouse within the window (normalized between 0 and 1) and uses that to set the targetFrequency variable, which determines the frequency of the generated sine wave.
The mouseDragged function is called whenever the mouse is moved while a button is held down. It calculates the horizontal position of the mouse within the window (normalized between 0 and 1) and uses that to set the pan variable.
The mousePressed function is called whenever a mouse button is pressed within the application window. It sets the bNoise variable to true, which causes the generator to output noise instead of a sine wave. It also loads an image file named "xingkong.jpg".
The mouseReleased function is called whenever a mouse button is released within the application window. It sets the bNoise variable to false, which causes the generator to output a sine wave again. It also loads an image file named "xingkong4.jpeg".
The audioOut function is called by the audio output system to generate the audio output buffer. It first calculates the scaling factors for the left and right channels based on the pan variable. If bNoise is true, it generates white noise for each sample in the buffer. If bNoise is false, it generates a sine wave for each sample in the buffer using the phase and phaseAdder variables, which are used to calculate the phase of the sine wave. The volume variable controls the overall volume of

the output, and the lAudio and rAudio arrays are used to store the audio samples for the left and right channels, respectively. The generated audio is written to the output buffer, which is passed to the function as a parameter.