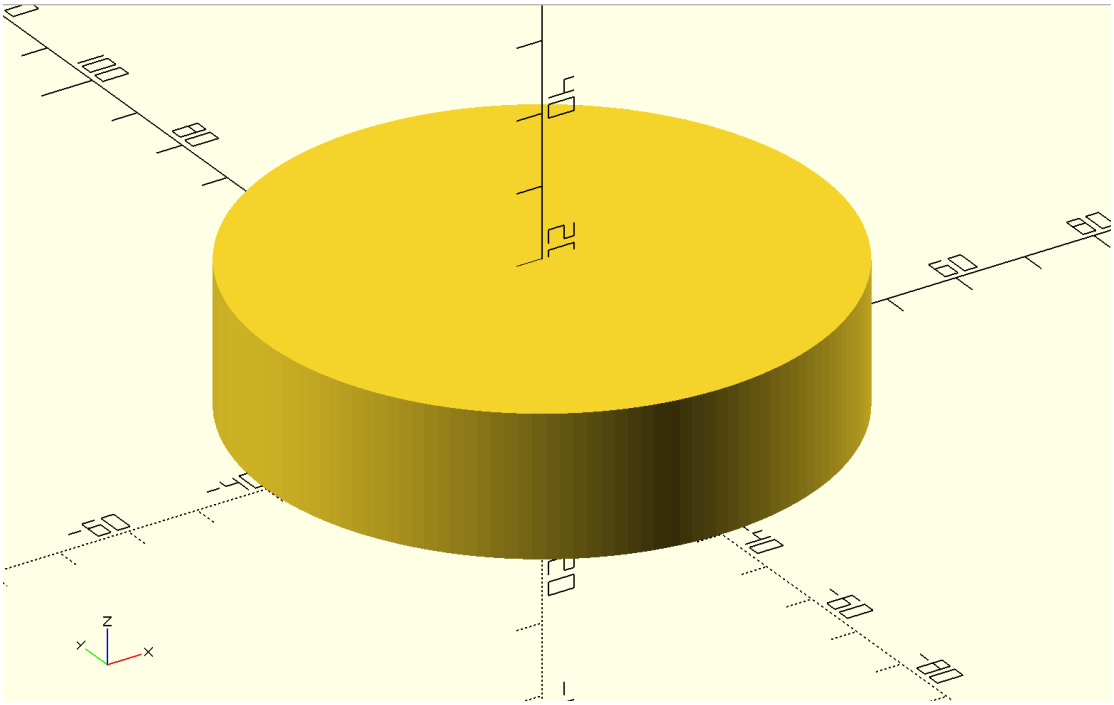
Final Report

Mooncake Making Steps Step 1: Create the Mooncake Body In this step, we will create the basic shape of the mooncake, which is a cylinder. Code

```
// Mooncake parameters
mooncake_radius = 40;  // Mooncake radius
mooncake_height = 20;  // Mooncake height
resolution = 150;      // Smoothness of the circle, the larger the value,
the smoother (segments)

// Create the mooncake body
module mooncake_body() {
    difference() {
        cylinder(r = mooncake_radius, h = mooncake_height, $fn =
resolution);
    }
}

// Render the mooncake body
mooncake_body();
```

Image



Step 2: Add Patterns

In this step, I will add vertical patterns to the mooncake body, making it more real

Code

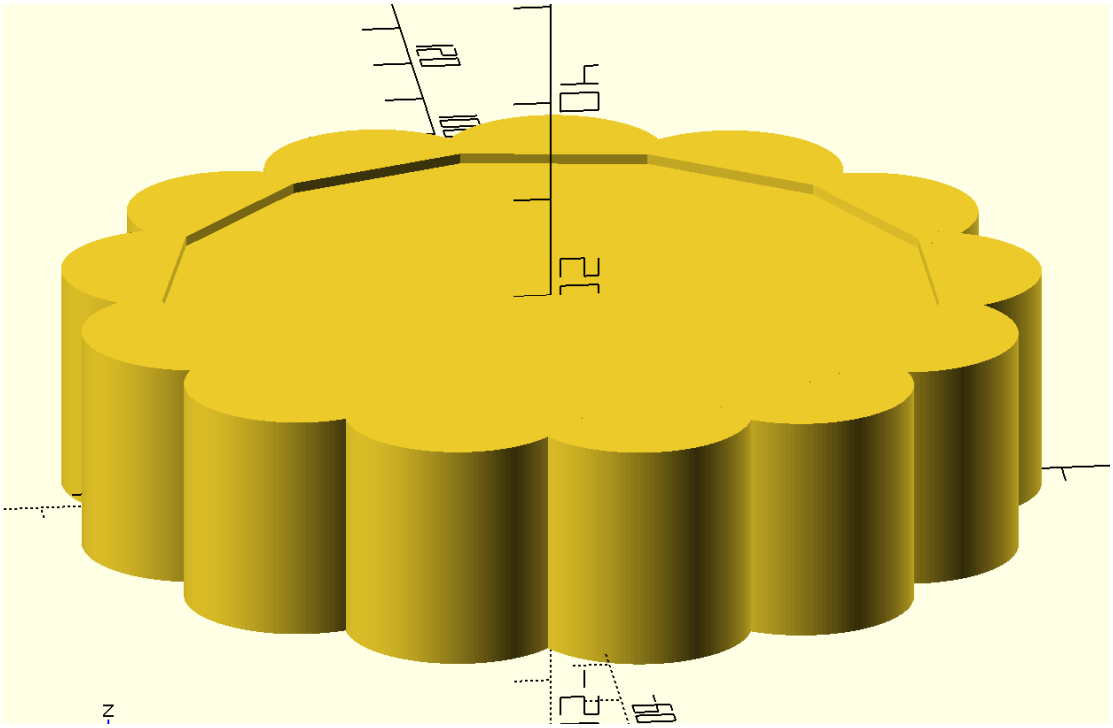
```
// Mooncake parameters
mooncake_radius = 40;  // Mooncake radius
mooncake_height = 20;  // Mooncake height
resolution = 150;      // Smoothness of the circle, the larger the value,
the smoother (segments)

// Pattern parameters
pattern_count = 13;    // Number of patterns
pattern_radius = 11;   // Radius of the pattern's semicircle
pattern_height = 0;    // Height of the pattern

module pattern(height) {
    translate([mooncake_radius-3, 0, height/2]) // Move to the middle of
the side
    rotate([180, 0, 90]) // Adjust rotation direction to extend along the
edge
    linear_extrude(height = pattern_radius * 2, center = true)
    difference() {
        circle(r = pattern_radius, $fn = resolution);
        translate([-pattern_radius, -pattern_radius * 2])
        square([pattern_radius * 2, pattern_radius * 2]);
    }
}
```

```
module mooncake_body() {
    cylinder(r = mooncake_radius, h = mooncake_height, $fn = resolution);

    // Add patterns, extending from bottom to top
    for (i = [0:pattern_count - 1]) {
        rotate([0, 0, i * 360 / pattern_count])
        pattern(20);
    }
}

// Render the mooncake body with patterns
mooncake_body();
```

Explanation

In this step, we define a function called pattern that creates a pattern on the side of the mooncake. We then use a loop to add multiple patterns around the mooncake body.

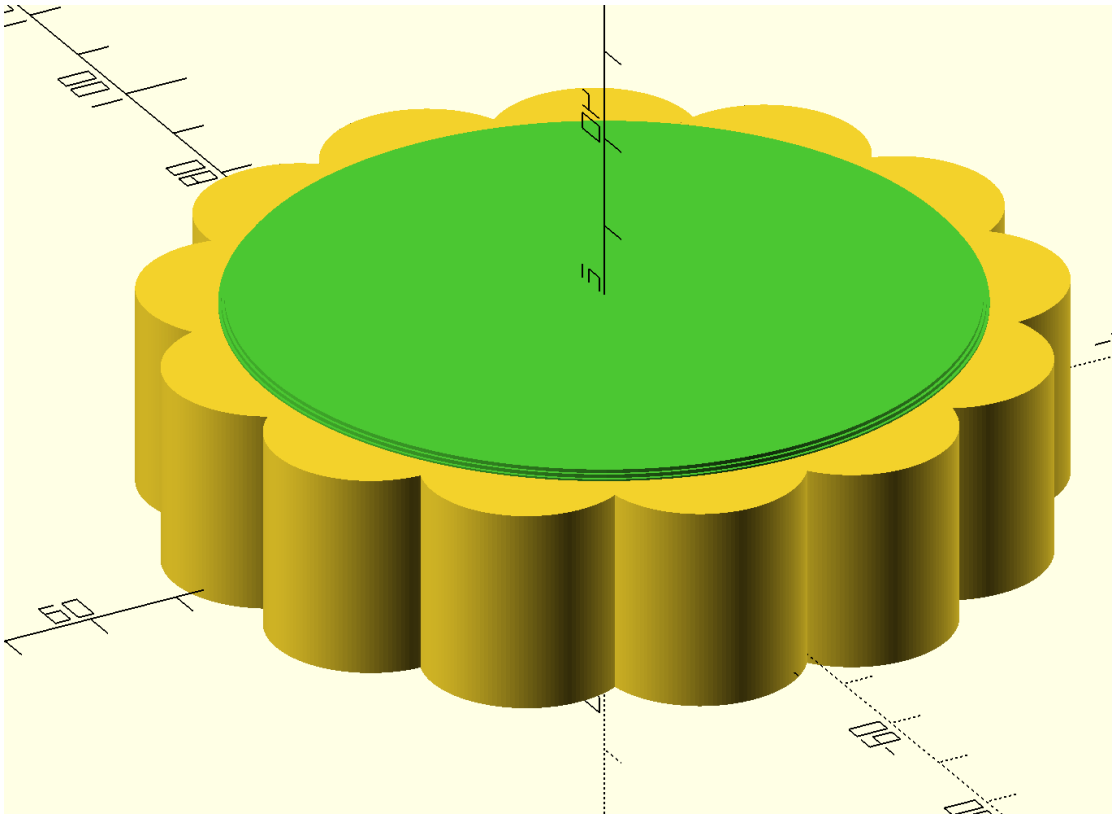
Image



Step 3: Adding Top Patten

In this process, I create a circular truncated cone at the top of the cylinder. This function is like a stander for subsequent patten on mooncake, including text and TaiJi patten.

```
// a fregment of about creating circular truncated cone
// Top bump parameter

top_height = 2;         // Total height of the top protrusion
top_layer_count = 5;    // Number of stepped layers
mooncake_radius = 40;   // Radius of the mooncake
resolution = 150;       // Smoothness of the circle, the larger the
smoother (segments)

module top_pattern() {
    for (i = [0:top_layer_count-1]) {
        layer_height = top_height / top_layer_count;
        layer_radius = mooncake_radius * (1 - i * 0.008);  // Radius
decreases with each layer

        translate([0, 0, i * layer_height])
        cylinder(r = layer_radius, h = layer_height, $fn = resolution);
    }
}
```

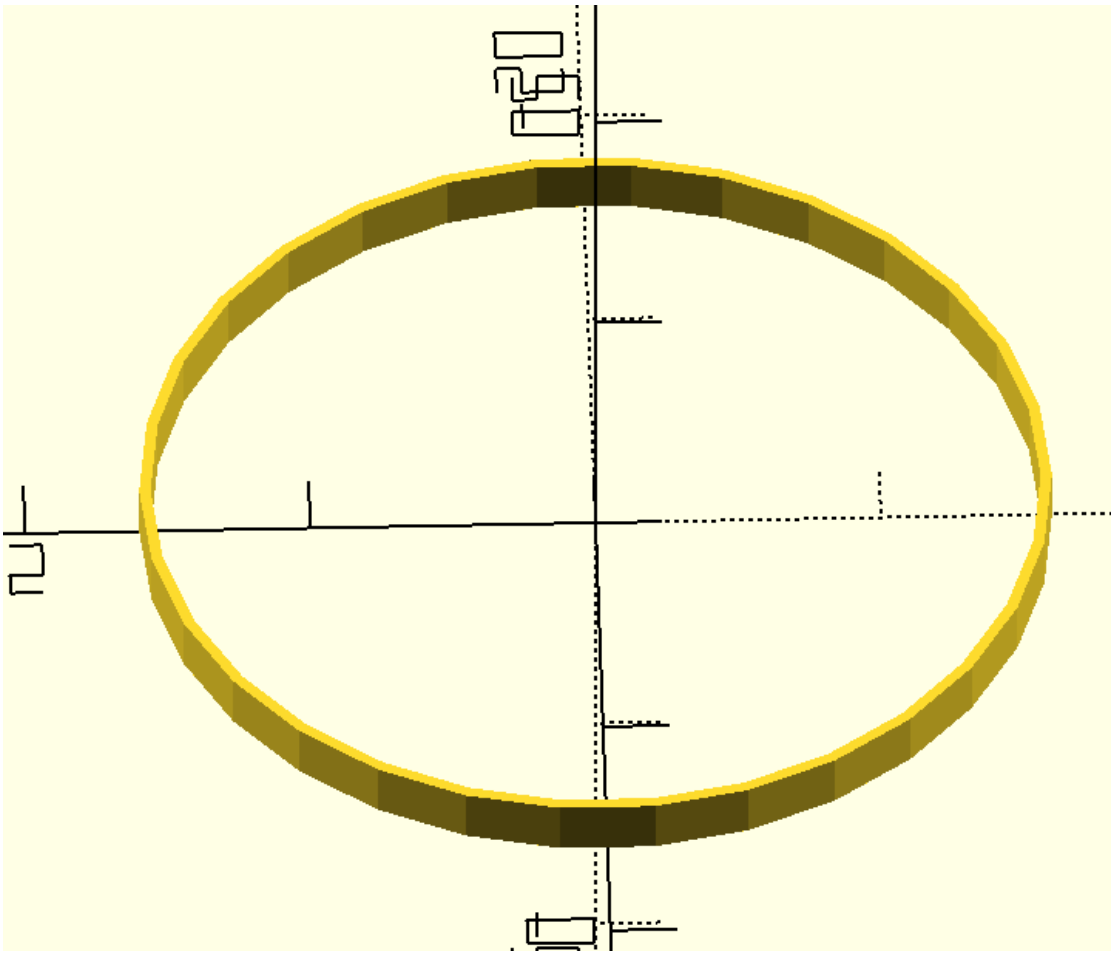


The green part is created by code.

Step 4: TaiJi patten

This is the most complicated part of this project. We need to use multiple Boolean operations such as difference, intersection The design of Tai chi patterns needs to go through the following stages:

```
module taiji() {
    r = mooncake_radius * 0.4;
    large_circle_radius = r;
    small_circle_radius = r / 2;

    linear_extrude(height = height) {
        // Draw the large outer circle
        difference(){
            circle(r = large_circle_radius);
            circle(r = large_circle_radius-0.4);
        }
}
```

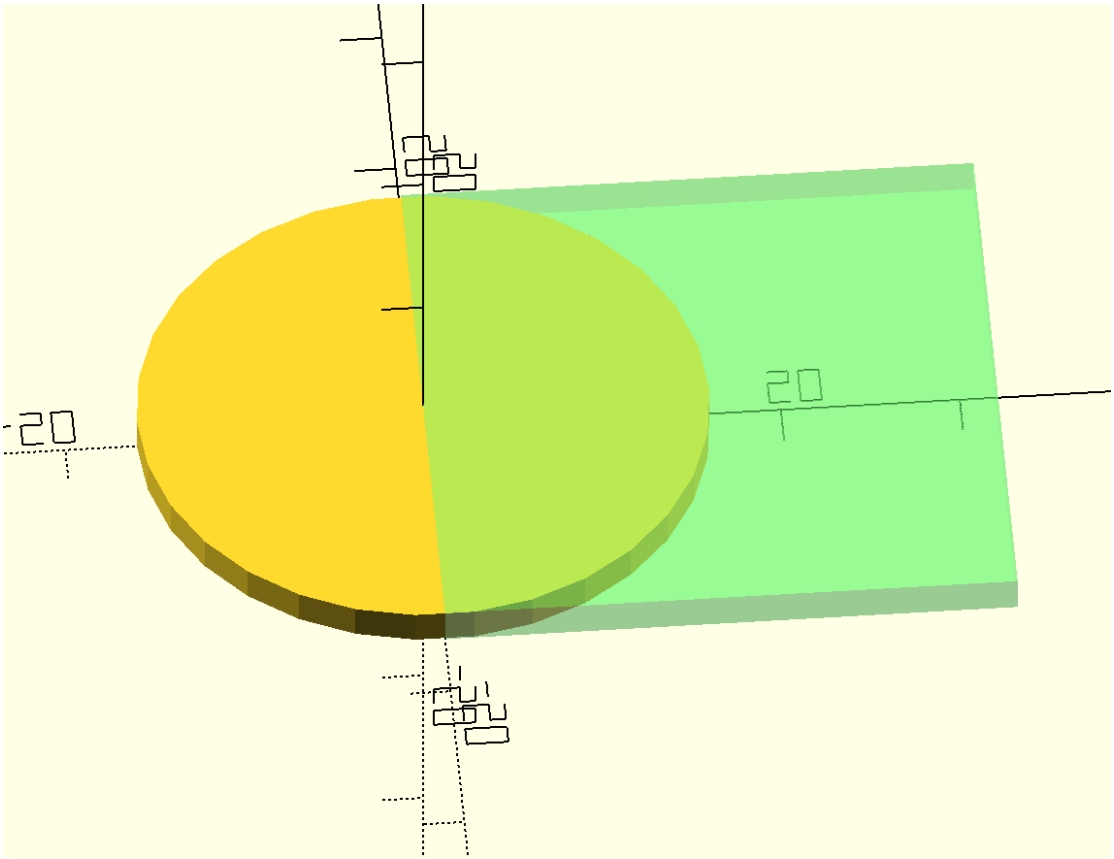


Seeing the patten of TaiJi, we can protrate that TaiJI symbol as: a union of a large circle and a semicircle, unioned with a small circle whose radius is half the radius of the large circle, and then differenced with the other half of the large circle. Thus, we can convert the task into three: 2.1: Creating the main half circle; 2.2 union or difference two small circle.

2.1 Main half circle

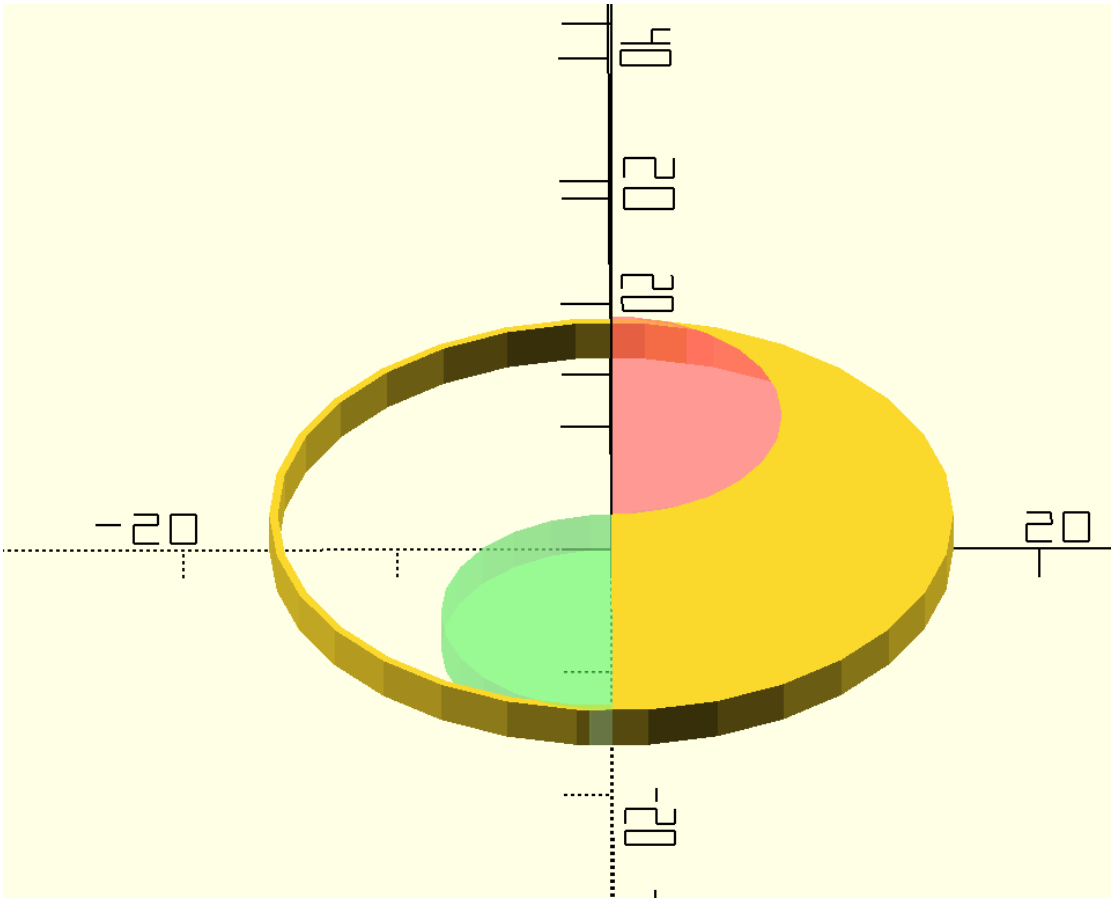
```
linear_extrude(height = height) { // converting 2D to 3D
        // Draw the large outer circle
        difference(){
            circle(r = large_circle_radius);
            circle(r = large_circle_radius-0.4);
        }

        difference() {
            union() {


                intersection() {
                    circle(r = large_circle_radius); // creating circle
                    translate([large_circle_radius, 0]) //moving position
```

```
                    square([large_circle_radius * 2, large_circle_radius *
2], center = true);
                }
            }
        }
    }
```



As code and graph demonstrate, the half circle is created by intersecting a circle and a square on circles' one side. 2.2 union or difference between two small circles. Similarly, we can create two little half circles, but one is Union to the main half circle another is Difference.

Thus, we can get the final image demonstration and code:




```
mooncake_radius = 40;
height = 2; // 添加高度参数，单位为毫米

module taiji() {
    r = mooncake_radius * 0.4; // 太极半径
    large_circle_radius = r;
    small_circle_radius = r / 2;

    linear_extrude(height = height) { // 添加拉伸操作
        // Draw the large outer circle
        difference(){
            circle(r = large_circle_radius);
            circle(r = large_circle_radius-0.4);
        }

        difference() {
            union() {
                // 左半圆
                translate([0, -small_circle_radius])
                intersection() {
                    circle(r = small_circle_radius);
```

```
                    translate([-small_circle_radius, 0])
                    square([small_circle_radius * 2, small_circle_radius *
2], center = true);
                }
                // 大圆的后半部分
                intersection() {
                    circle(r = large_circle_radius);
                    translate([large_circle_radius, 0])
                    square([large_circle_radius * 2, large_circle_radius *
2], center = true);
                }
            }
            // 右半圆的后半部分
            translate([0, small_circle_radius])
            intersection() {
                circle(r = small_circle_radius);
                translate([small_circle_radius, 0])
                square([small_circle_radius * 2, small_circle_radius * 2],
center = true);
            }
        }
    }
}

taiji();
```

Step 4: Finish Up

After adding some text on it, we can get the finally code.

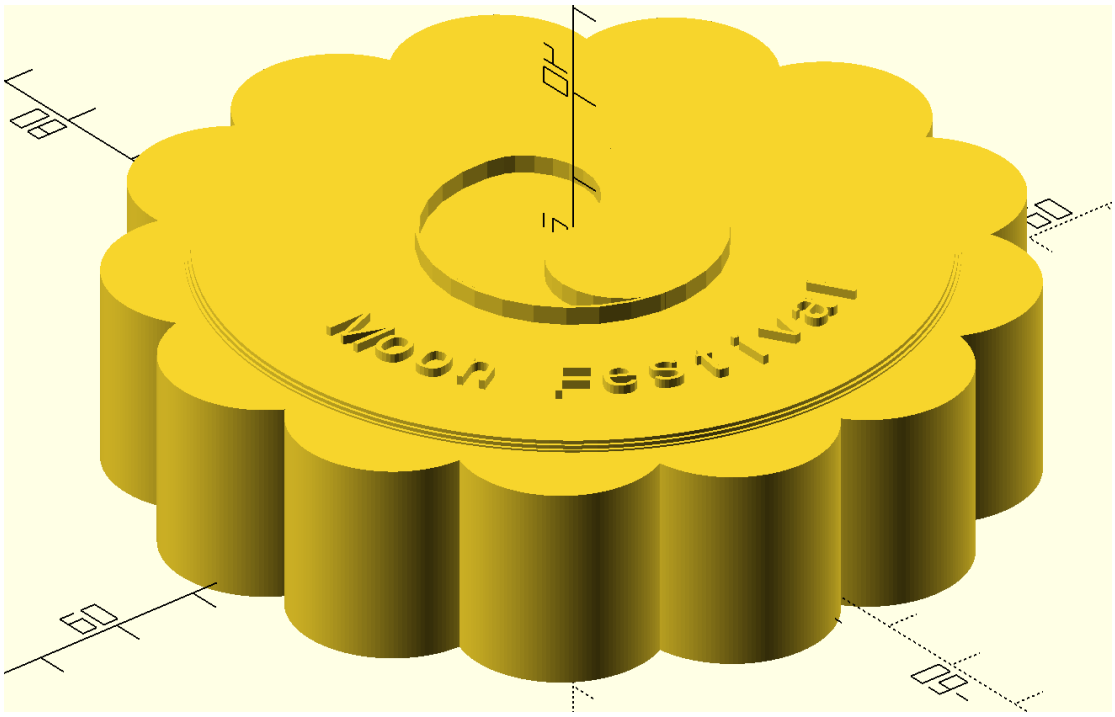
```
// Mooncake parameters
mooncake_radius = 40;  // Mooncake radius
mooncake_height = 20;  // Mooncake height
resolution = 150;      // Smoothness of the circle, the larger the
smoother (segments)

// Pattern parameters
pattern_count = 13;    // Number of patterns
pattern_radius = 11;   // Radius of the pattern semicircle
pattern_height = 0;    // Height of the pattern protrusion

// Top protrusion parameters
top_height = 2;        // Total height of the top protrusion
top_layer_count = 5;   // Number of steps

// Taiji parameters
height_taiji = 2;      // Taiji height

module pattern(height) {
    translate([mooncake_radius - 3, 0, height / 2]) // Move to the middle
```

```
of the side
    rotate([180, 0, 90]) // Adjust the rotation direction to extend along
the edge
    linear_extrude(height = pattern_radius * 2, center = true)
    difference() {
        circle(r = pattern_radius, $fn = resolution);
        translate([-pattern_radius, -pattern_radius * 2])
        square([pattern_radius * 2, pattern_radius * 2]);
    }
}

module top_pattern() {
    for (i = [0:top_layer_count - 1]) {
        layer_height = top_height / top_layer_count;
        layer_radius = mooncake_radius * (1 - i * 0.008);  // Decrease the
radius of each layer

        translate([0, 0, mooncake_height + i * layer_height])
        cylinder(r = layer_radius, h = layer_height, $fn = resolution);
    }
}

module taiji() {
    r = mooncake_radius * 0.4; // Taiji radius
    large_circle_radius = r;
    small_circle_radius = r / 2;

    translate([0, 0, mooncake_height + top_height + 0.1])
    rotate([0, 0, 270])

    linear_extrude(height = height_taiji) { // Add extrusion operation
        // Draw the large outer circle
        difference() {
            circle(r = large_circle_radius);
            circle(r = large_circle_radius - 0.4);
        }

        difference() {
            union() {
                // Left semicircle
                translate([0, -small_circle_radius])
                intersection() {
                    circle(r = small_circle_radius);
                    translate([-small_circle_radius, 0])
                    square([small_circle_radius * 2, small_circle_radius *
2], center = true);
                }
                // Rear half of the large circle
                intersection() {
                    circle(r = large_circle_radius);
                    translate([large_circle_radius, 0])
                    square([large_circle_radius * 2, large_circle_radius *
2], center = true);
                }
```

```
            }
            // Rear half of the right semicircle
            translate([0, small_circle_radius])
            intersection() {
                circle(r = small_circle_radius);
                translate([small_circle_radius, 0])
                square([small_circle_radius * 2, small_circle_radius * 2],
center = true);
            }
        }
    }
}

module text_pattern() {
    text_str = "Moon Festival";  // Change to the English name of the Mid-
Autumn Festival
    char_count = len(text_str);
    font_size = mooncake_radius * 0.13; // Reduce the font size to fit the
circumference
    extrude_height = 3.0; // Height of the text protrusion

    // Arrange the text around the top of the mooncake in a circle
    for (i = [0:char_count - 1]) {
        angle = i * 130 / char_count;
        char = text_str[i];

        rotate([0, 0, angle])
        translate([0, mooncake_radius * 0.7, mooncake_height]) // Place on
top of the mooncake
        rotate([0, 0, 180]) // Rotate the text to be readable in top view
        linear_extrude(height = extrude_height)
        text(str(char), size = font_size, halign = "center", valign =
"center");
    }
}

module mooncake_body() {
    difference() {
        cylinder(r = mooncake_radius, h = mooncake_height, $fn =
resolution);

        // Add a small chamfer at the bottom
        //translate([0, 0, -0.1])
        //cylinder(r1 = mooncake_radius + 1, r2 = mooncake_radius - 1, h =
2, $fn = resolution);
    }

    // Add patterns extending from the bottom to the top
    for (i = [0:pattern_count - 1]) {
        rotate([0, 0, i * 360 / pattern_count])
        pattern(20);
    }

    // Add top protrusion
```

```
    top_pattern();

    // Add top decorative pattern
    taiji();

    // Add "Moon Festival" text
    text_pattern();
}

mooncake_body(); // Render the mooncake body
```

After render, we can get this image about mooncake:



Author(both for this document and model creating): Tony Wayne Wang

Rendering

Step 1: Create project framework

In order to put the models into an animated, rendered 3D scene, the WebGL-based library, THREE.js was used. To start the project, first I composed a simple HTML file that loads in the main THREE.js javascript codes from the file main.js, so that it may be displayed on a browser page:

index.html > html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Three.js STL Loader</title>
</head>
<body>
    <script type="module" src="main.js"></script>
</body>
</html>
```

Step 2: THREE.js scene initialization

As THREE.js is an external 3D library for the coding language javascript, we start first by creating a javascript file and importing the necessary libraries:

```javascript
import * as THREE from 'https://esm.sh/three@0.155.0';
import { STLLoader } from 'https://esm.sh/three@0.155.0/examples/jsm/loaders/STLLoader.js';
import { OrbitControls } from 'https://esm.sh/three@0.155.0/examples/jsm/controls/OrbitControls.js';
import { GUI } from 'https://esm.sh/lil-gui';
```

Then, we initialize essential components for our rendering, which include the scene, renderer, ambient/point lighting, variables, and texture resources. A few lines of the texture initialization can be seen marked off as comments as the resource they load in are UV checkers that won't be present in the final rendering.

```
//Scene initialization
const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera(75,window.innerWidth / window.innerHeight,0.1,10000);
camera.position.z = 10;
const spinParams = {distance : 10, speed : 100}

const renderer = new THREE.WebGLRenderer({ antialias: true });
renderer.setSize(window.innerWidth, window.innerHeight);
renderer.setClearColor(0xeeeeee);
renderer.shadowMap.enabled = true;
renderer.shadowMap.type = THREE.PCFSoftShadowMap;
document.body.appendChild(renderer.domElement);

//Loading textures & using one as the scene background
const fullMoonNightTexture = new THREE.TextureLoader().load('assets/texture_FullMoon.jpeg');
const tableTexture = new THREE.TextureLoader().load('assets/texture_Wood.jpg');
//const uvCheckerTexture = new THREE.TextureLoader().load('assets/texture_Checker.jpg');
// const flatColorTexture = new THREE.TextureLoader().load('assets/texture_FlatColor.png');
// const gradientTexture = new THREE.TextureLoader().load('assets/texture_Gradient.png');
// const otherGradientTexture = new THREE.TextureLoader().load('assets/texture_OtherGradient.png');
scene.background = fullMoonNightTexture;

//Ambient Lighting
scene.add(new THREE.AmbientLight(0x36354d, 2)); //Ambient Lighting

//Spotlight initialization
const spotlight = new THREE.SpotLight(0xe09322, 5);
spotlight.position.set(5, 10, 5);
spotlight.angle = Math.PI / 6;
spotlight.penumbra = 0.3;
spotlight.castShadow = true;
spotlight.intensity = 200;
spotlight.shadow.mapSize.width = 2048;
spotlight.shadow.mapSize.height = 2048;
scene.add(spotlight);
```

UV Checker Texture example

Initialized scene without models

Step 3: Adding models/geometries to the new scene

Now with the scene initialized, we write code to add 3D objects into the scene. The code below showcases the initialization of both the table and 3 mooncakes.

-Notice that the table's initialization did not involve using STLLoader, that's because the THREE.js library offers built-in functions for initializing simple geometries, which is enough to support adding a box-shaped table.

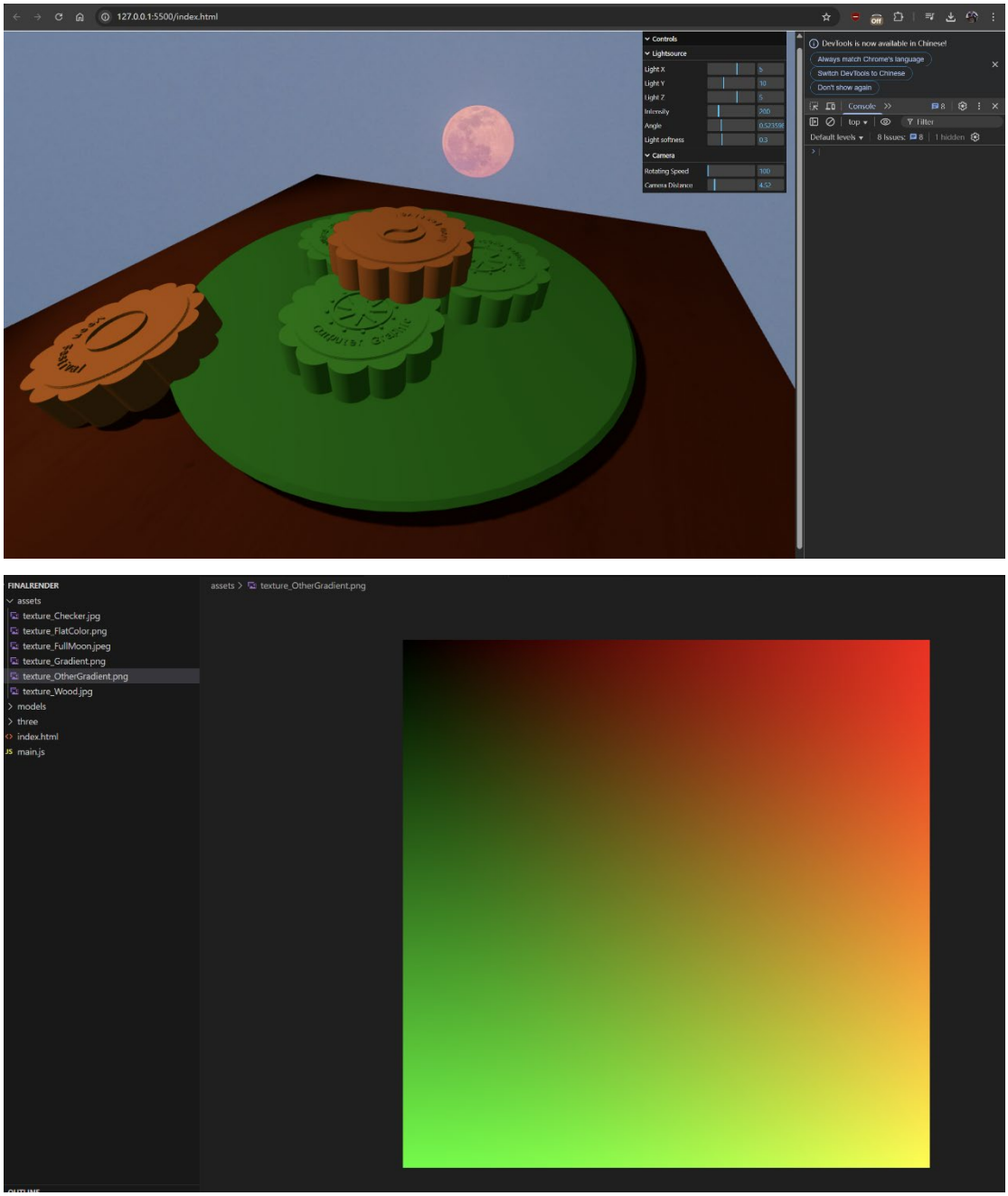
-For the code initializing the mooncake models, a for loop is used to load in multiple of the same model while keeping the code compact.

-The mooncakes' material does not have a texture map due the STL models not having working UVs. Efforts to apply the texture checker to the models reveal that they seem to be only reading a single pixel from the textures, which give the models a solid color instead of any desired patterns.

```
// Table geometry
const tableGeometry = new THREE.BoxGeometry(10, 0.5, 10);
const tableMaterial = new THREE.MeshStandardMaterial({ color: 0x7a3d00, map: tableTexture });
const table = new THREE.Mesh(tableGeometry, tableMaterial);
table.position.set(0, -0.25, 0);
table.receiveShadow = true;
scene.add(table);

//Load STL models
let cakeMeshes = new Array(3);
const loader = new STLLoader();
const modelParamOne = { x: 1.2, y: 0.25, z: 0, rotateY: 0, rotateX: degToRad(-90), rotateZ: degToRad(240), rotating: false };
const modelParamTwo = { x: -0.6, y: 0.25, z: Math.sqrt(3)*.6, rotateY: 0, rotateX: degToRad(-90), rotateZ: degToRad(120), rotating: false };
const modelParamThree = { x: -0.6, y: 0.25, z: -Math.sqrt(3)*.6, rotateY: 0, rotateX: degToRad(-90), rotateZ: 0, rotating: false };
const modelParams = [modelParamOne,modelParamTwo,modelParamThree];
//Load in 3 mooncakes
for (let i=0; i < 3; i++){
    loader.load('models/v1.stl', function (geometry) {
        geometry.computeBoundingBox();
        const size = geometry.boundingBox.getSize(new THREE.Vector3());
        const scaleFactor = 2 / Math.max(size.x, size.y, size.z);
        const center = geometry.boundingBox.getCenter(new THREE.Vector3());

        const material = new THREE.MeshStandardMaterial({
            // map: uvCheckerTexture,
            color: 0xa86002,
            metalness: 0.005,
            roughness: 0.5
        });

        cakeMeshes[i] = new THREE.Mesh(geometry, material);
        cakeMeshes[i].scale.set(scaleFactor, scaleFactor, scaleFactor);
        cakeMeshes[i].position.set(-center.x * scaleFactor, 0.5, -center.z * scaleFactor);
        cakeMeshes[i].castShadow = true;
        scene.add(cakeMeshes[i]);

        cakeMeshes[i].position.set(modelParams[i].x, modelParams[i].y, modelParams[i].z);
        cakeMeshes[i].rotation.set(modelParams[i].rotateX, modelParams[i].rotateY, modelParams[i].rotateZ);

    });
}
```

```
//Load in the plate
let plateMesh;
loader.load('models/plate.stl', function (geometry) {
    geometry.computeBoundingBox();
    const size = geometry.boundingBox.getSize(new THREE.Vector3());
    const scaleFactor = 2 / Math.max(size.x, size.y, size.z);
    const center = geometry.boundingBox.getCenter(new THREE.Vector3());

    const material = new THREE.MeshStandardMaterial({
        // map: uvCheckerTexture,
        color: 0xFFFFFF,
        metalness: 0.3,
        roughness: 0.2
    });

    plateMesh = new THREE.Mesh(geometry, material);
    plateMesh.scale.set(scaleFactor, scaleFactor, scaleFactor);
    plateMesh.position.set(-center.x * scaleFactor, 0.5, -center.z * scaleFactor);
    plateMesh.castShadow = true;
    scene.add(plateMesh);

    plateMesh.position.set(0, -.02, 0);
    plateMesh.rotation.set(degToRad(-90),0,0);
    plateMesh.scale.set(.08,.08,.027);

});
```

Loading in the plate, scale manipulation is used on top of position and rotation to make the plate the desired size, eliminating the need of doing so in the modeling process.



Example of a gradient texture being applied to the STL models. As it shows here, the pattern is not visible due to the models' UVs not supporting it.

Scene with models loaded

Step 4: Adding Particle Effect and its GUI

```
function createParticles() {
    //Remove existing particles
    if (particles) {
        scene.remove(particles);
        particleGeometry.dispose();
        particleMaterial.dispose();
    }

    const positions = [];
    basePositions = [];

    for (let i = 0; i < particleParams.count; i++) {
        const x = (Math.random() - 0.5) * 10;
        const y = (Math.random() - 0.5) * 5 + 2; // Add y-axis offset
        const z = (Math.random() - 0.5) * 10;
        positions.push(x, y, z);
        basePositions.push({ x, y, z });
    }

    particleGeometry = new THREE.BufferGeometry();
    particleGeometry.setAttribute('position', new THREE.Float32BufferAttribute(positions, 3));

    const particleTexture = new THREE.TextureLoader().load('assets/particle_star.png');
    particleTexture.colorSpace = THREE.SRGBColorSpace;

    particleMaterial = new THREE.PointsMaterial({
        size: particleParams.size,
        sizeAttenuation: true,
        map: particleTexture,
        alphaTest: 0.5,
        transparent: true,
        depthWrite: false,
        blending: THREE.AdditiveBlending,
        opacity: 0.8
    });

    particleMaterial.color.setHSL(0.05, 1.0, 0.6);

    particles = new THREE.Points(particleGeometry, particleMaterial);
    scene.add(particles);
}
```

This function load the particle assets so that they popped in the scene with certain parameters like popping position and their changing height to simulate floating movement.

```
const particleFolder = gui.addFolder('Particles');
particleFolder.add(particleParams, 'count', 100, 5000, 100).name("Amount").onChange(createParticles);
particleFolder.add(particleParams, 'size', 0.05, 1).name("Size").onChange(() => {
    particleMaterial.size = particleParams.size;
});
particleFolder.add(particleParams, 'speed', 0.1, 5).name("Speed");
particleFolder.open();
```

This block of code is for creating the GUI that let users play with the particles like changing their floating speed, density and size.

Step 5: Making the turntable animation & other features

To animate the scene, a simple function rotate Cam() is written to change the camera's X and Z position in relation to the sine and cosine of the current time, effectively creating a turntable effect.

```
//Camera rotation, used modified Saty's code
function rotateCam(){
    const timer = Date.now()*(1/(2050-spinParams.speed));
    camera.position.x = Math.cos(timer)*spinParams.distance;
    camera.position.z = Math.sin(timer)*spinParams.distance;

    renderer.render(scene, camera);
}

//Interactions
const controls = new OrbitControls(camera, renderer.domElement);
controls.enableDamping = true;

//Window size monitor
window.addEventListener('resize', () => {
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();
    renderer.setSize(window.innerWidth, window.innerHeight);
});
```

Then, this function is added to the animate() function, executed every update:

```
//Animation
function animate() {
    requestAnimationFrame(animate);
    rotateCam();
    controls.update();
}
animate();
```

Lastly, a GUI for the user to change the scene parameters on the fly is added, making the scene interactable and allowing for dynamic manipulation of its presentation:

```
//GUI for components
const gui = new GUI();
const lightFolder = gui.addFolder('Lightsource');
lightFolder.add(spotlight.position, 'x', -20, 20).name("Light X");
lightFolder.add(spotlight.position, 'y', 0, 30).name("Light Y");
lightFolder.add(spotlight.position, 'z', -20, 20).name("Light Z");
lightFolder.add(spotlight, 'intensity', 50, 700).name("Intensity");
lightFolder.add(spotlight, 'angle', 0.1, Math.PI / 2).name("Angle").onChange(() => spotlight.updateMatrix());
lightFolder.add(spotlight, 'penumbra', 0, 1).name("Light softness");
lightFolder.open();

const camFolder = gui.addFolder('Camera');
camFolder.add(spinParams, 'speed', 100,2000).name('Rotating Speed');
camFolder.add(spinParams, 'distance', 2,20).name('Camera Distance');
camFolder.open();
```

The scene at this stage presents as follows, with parameters altered using the GUI controls: