G5_Final_report

Project Background:

Chinese pastry is an important part of China's food culture, with a wide variety of unique flavors, and is usually served as a snack at breakfast, afternoon tea or banquets. Pastry is not only delicious, but also reflects the culinary characteristics and cooking skills of different parts of China.

Chinese pastry is not only a kind of food, but also a cultural symbol, reflecting the Chinese people's love of life and exquisite pursuit of food. Whether it is street food or high-class banquets, dim sum is loved by people for its unique flavor and artistry.

Our project aims to create 3DCG content of Chinese pastry (Water Lily Crisp, Moon Cake, Sweetheart Pastry, Peach Blossom Pastry) through OpenSCAD and Three.js. Through this project, we hope to contribute to the preservation of traditional Chinese food!

Topic Analysis:

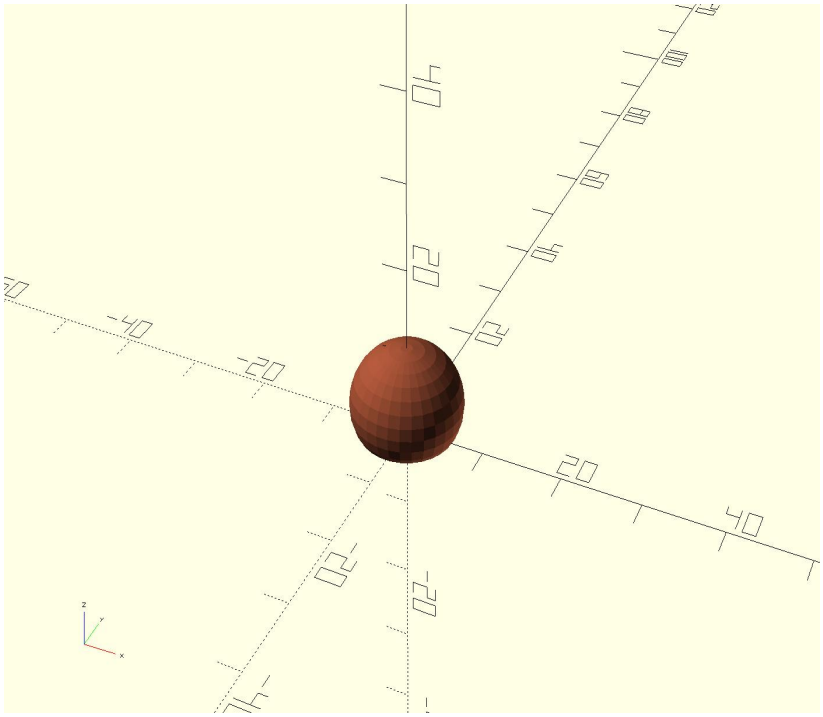
Modelling

Phase 1:

We used OpenSCAD to create initial models of each of the four different snacks, which provided the basis for the subsequent steps. These models contained the characteristics of the snacks, including the shape, pattern, and sesame seed embellishment. We even created boxes for these treats. All of our work will be based on and developed from these basic models.

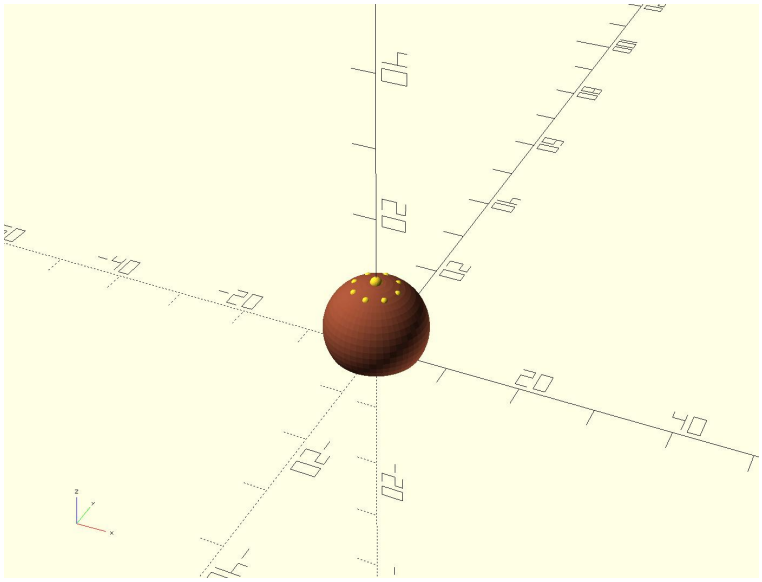
Zhengxuan Li's model

1.Use the sphere minus the bottom to form the filling for the pastry.



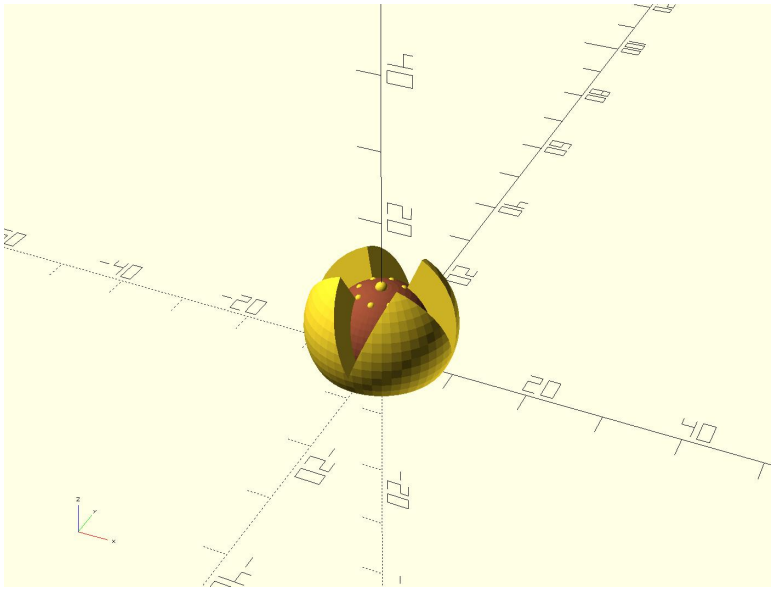
```
color([0.6,0.3,0.2])
translate([0,0,0.1])
scale([0.7,0.7,0.7]){
    difference(){
        translate([0,0,5])
        sphere(r=10);
        translate([0,0,-10])
        cube(20,center=true);
    }
}
translate([0,0,10.5])
sphere(r=0.8);
```

2.Sprinkle some garnishes on top of the filling to give it a more sophisticated look.



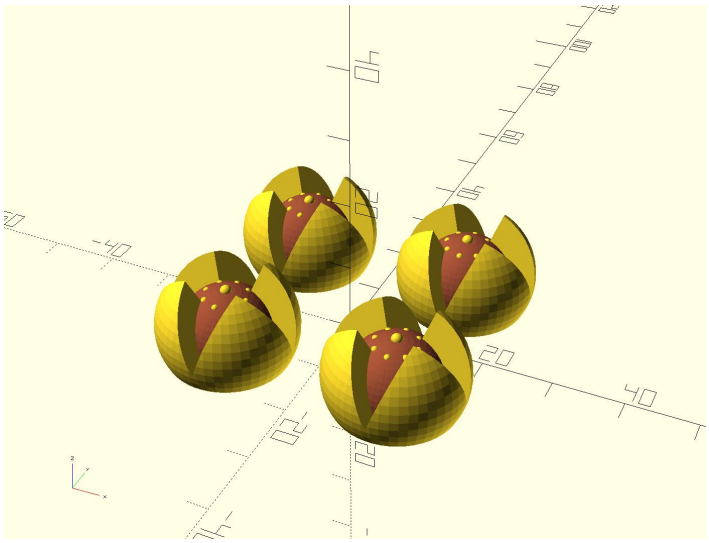
```
module dot(){
sphere(r=0.5);
}
r=3;
n=8;
step=360/n;

for(i=[0:step:365]){
    angle=i;
    dx=r*cos(angle);
    dy=r*sin(angle);
    translate([dx,dy,9.6])
    dot();
}
```

3.Use Boolean operations to add snack cracked puff pastry.



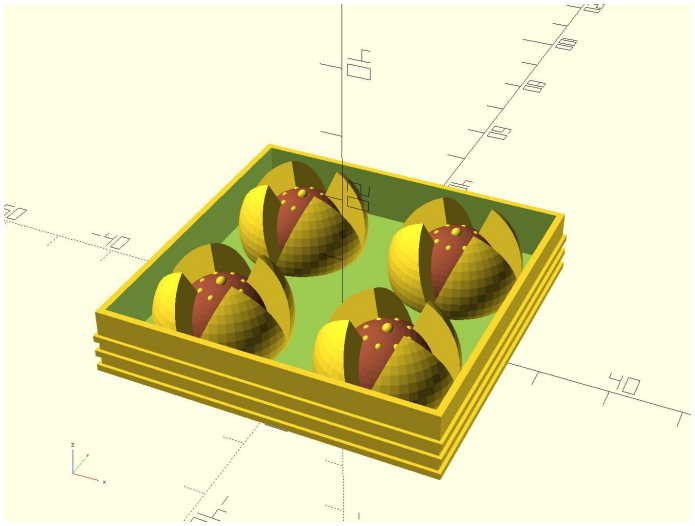
```
difference(){
    difference(){
        difference(){
            translate([0,0,5])
            sphere(r=10);
            translate([0,0,-10])
            cube(20,center=true);
        }
        scale([1.1,1.1,1.1]){
            difference(){
                    difference(){
                        translate([0,0,5])
                        sphere(r=10);
                        translate([0,0,-10])
                        cube(20,center=true);
                    }
                    translate([0,15,0])
                    rotate([-20,0,0])
                    cube([30,30,60],center=true)

                    translate([0,-15,0])
                    rotate([20,0,0])
                    cube([30,30,60],center=true);
                }
                rotate([0,0,90]){
                    difference(){
                        difference(){
                                translate([0,0,5])
                                sphere(r=10);

                                translate([0,0,-10])
                                cube(20,center=true);
                        }
                        translate([0,15,0])
                        rotate([-20,0,0])
                        cube([30,30,60],center=true);

                        translate([0,-15,0])
                        rotate([20,0,0])
                        cube([30,30,60],center=true);
                }
            }
        }
    }
}
```

4.Generate multiple snacks via module() and loops.



```
translate([-12.5,-12.5,0])
for (dx=[0,25]){
    for (dy=[0,25]){
        translate([dx,dy,0])
        dessert();
    }
}
```

5. Add a box for four snacks.



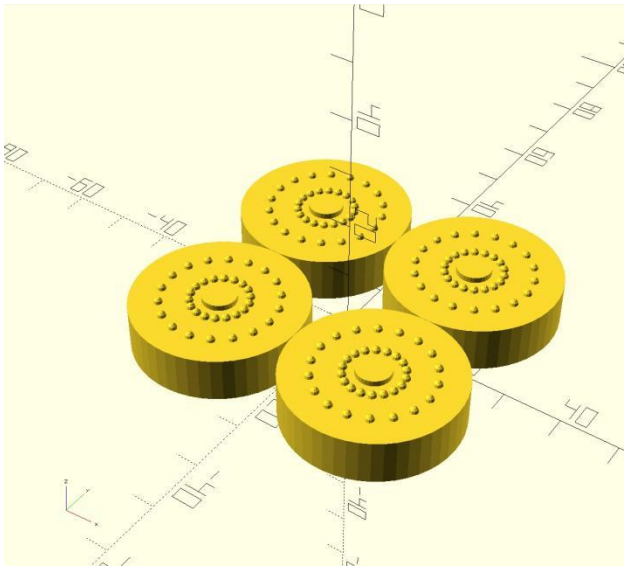
```
translate([0,0,4.9])
difference(){
    cube([50,50,10],center=true);
    translate([0,0,2])
    cube([48,48,10],center=true);
}

module curb(){
    translate([0,0,0.5])
    difference(){
        cube([51,51,1],center=true);
        cube([50,50,2],center=true);
    }
}

for(dz=[0,2.5,5])
    translate([0,0,dz])
    curb();
```

Weiyu Ouyang's model

1. Create a cylinder as the body of the snack.

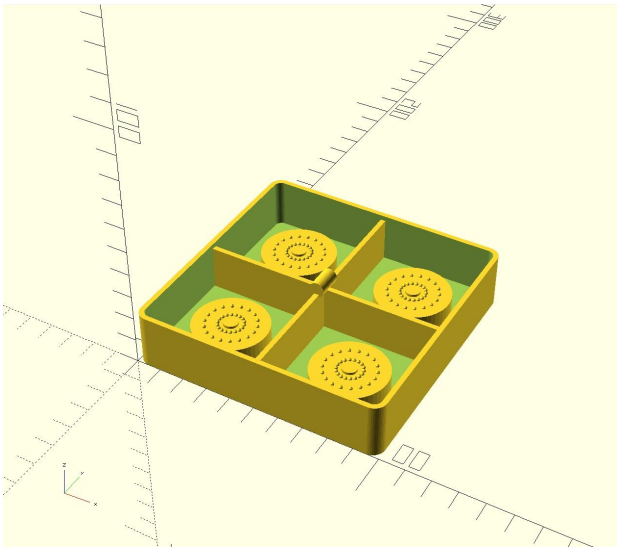
```
// 老婆饼的尺寸
pastry_radius = 15; // 老婆饼半径
pastry_height = 8;  // 老婆饼高度

// 创建老婆饼（仅基本形状，扁平的圆柱体）
module create_pastry() {
    // 饼的主体（扁平的圆柱体）
    cylinder(h=pastry_height, r=pastry_radius, $fn=50);
}

// 放置四块老婆饼
module place_pastries() {
    // 1
    translate([-pastry_radius, -pastry_radius, 0]) {
        create_pastry();
    }
    // 2
    translate([pastry_radius, -pastry_radius, 0]) {
        create_pastry();
    }
    // 3
    translate([-pastry_radius, pastry_radius, 0]) {
        create_pastry();
    }
    // 4
    translate([pastry_radius, pastry_radius, 0]) {
        create_pastry();
    }
}

// 生成老婆饼
place_pastries();
```

2. Add polka dots to the top for pattern and sesame seeds.



```
// 饼的主体（扁平的圆柱体）
cylinder(h=pastry_height, r=pastry_radius, $fn=50);

// 饼的边缘褶皱
for (i = [0:30:360]) {
    rotate([0, 0, i]) {
        translate([pastry_radius - 2, 0, 0]) {
            scale([1, 0.5, 1]) {
                cylinder(h=pastry_height, r=1.5, $fn=20);
            }
        }
    }
}

// 饼的表面花纹（模拟芝麻）
for (i = [0:10:360]) {
    rotate([0, 0, i]) {
        for (j = [5:5:pastry_radius - 2]) {
            translate([j * cos(i), j * sin(i), pastry_height]) {
                sphere(r=0.8, $fn=10); // 芝麻点
            }
        }
    }
}

// 饼的中心装饰（模拟花纹）
translate([0, 0, pastry_height]) {
    linear_extrude(height = 1) {
        circle(r=3, $fn=30); // 中心小圆点
```

3. Make the box.

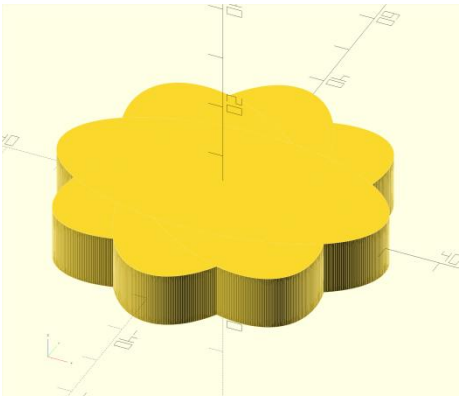


```
// 创建带圆角的盒子
module create_rounded_box(length, width, height, radius) {
    // 底部
    hull() {
        translate([radius, radius, 0]) {
            cylinder(h=height, r=radius, $fn=50);
        }
        translate([length - radius, radius, 0]) {
            cylinder(h=height, r=radius, $fn=50);
        }
        translate([radius, width - radius, 0]) {
            cylinder(h=height, r=radius, $fn=50);
        }
        translate([length - radius, width - radius, 0]) {
            cylinder(h=height, r=radius, $fn=50);
        }
    }
}

// 创建盒子
module create_box() {
    difference() {
        // 外盒
        create_rounded_box(box_length, box_width, box_height, corner_radius);
        // 内盒挖空
        translate([wall_thickness, wall_thickness, wall_thickness]) {
            create_rounded_box(box_length - 2*wall_thickness, box_width - 2*wall_thickness, box_height, corner_radius -
    wall_thickness);
        }
    }
}

// 创建隔板（十字形）
module create_dividers() {
    // 横向隔板
    translate([0, box_width/2 - divider_thickness/2, 0]) {
        cube([box_length, divider_thickness, box_height]);
    }
    // 纵向隔板
    translate([box_length/2 - divider_thickness/2, 0, 0]) {
        cube([divider_thickness, box_width, box_height]);
    }
}
```

```
// 创建盖子
module create_lid() {
    translate([0, 0, box_height]) {
        difference() {
            // 盖子外盒
            create_rounded_box(box_length, box_width, wall_thickness, corner_radius);
            // 盖子内盒挖空
            translate([wall_thickness, wall_thickness, -0.1]) {
                create_rounded_box(box_length - 2*wall_thickness, box_width - 2*wall_thickness, wall_thickness + 0.2,
    corner_radius - wall_thickness);
            }
        }
    }
}

// 创建把手
module create_handle() {
    translate([box_length / 2, box_width / 2, box_height + wall_thickness]) {
        rotate([90, 0, 0]) {
            cylinder(h=10, r=3, $fn=50, center=true);
        }
    }
}
```

Youyou Wu's model

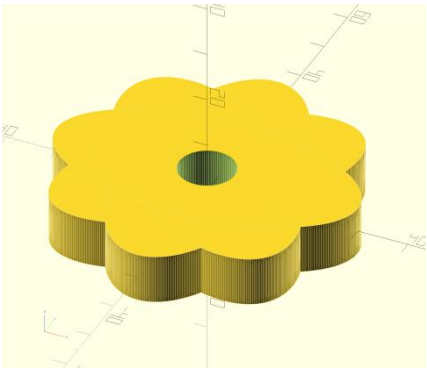
1. Simulate the shape of pastry petals.



```
petals = 8;
radius = 30;
thickness = 9.9;
height = 10;
layer_count = 1;
```

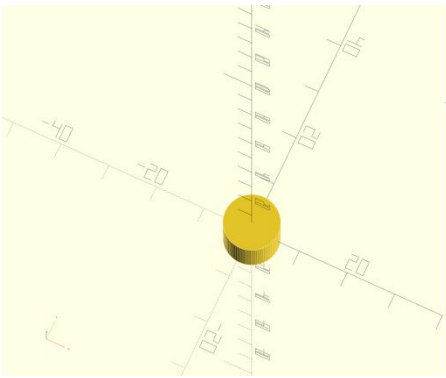
```
module jujube_flower_pastry() {
difference() {
for (i = [0:layer_count - 1]) {
translate([0, 0, i * 0.001]) {
for (j = [0:petals - 1]) {
rotate([0, 0, j * (360 / petals)]) {
petal();
}
}
}
}
cylinder(h=height, r=hole_radius, center=true);
}
}

jujube_flower_pastry();
```

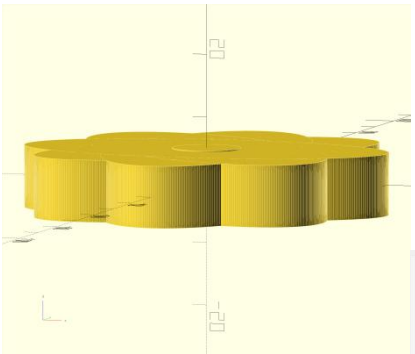
2. Leave the filling in the middle of the model



```
hole_radius = 5.5;


module petal() {
rotate([0, 0, 45.001]) {
resize([radius * 2, radius, thickness]) {
cylinder(h=thickness, r=radius, center=true);
}
}
}

module jujube_flower_pastry() {
difference() {
for (i = [0:layer_count - 1]) {
translate([0, 0, i * 0.001]) {
for (j = [0:petals - 1]) {
rotate([0, 0, j * (360 / petals)]) {
petal();
}
}
}
}
cylinder(h=height, r=hole_radius, center=true);
}
}

jujube_flower_pastry();
```

3. Use the sphere minus the filling of the pastry.

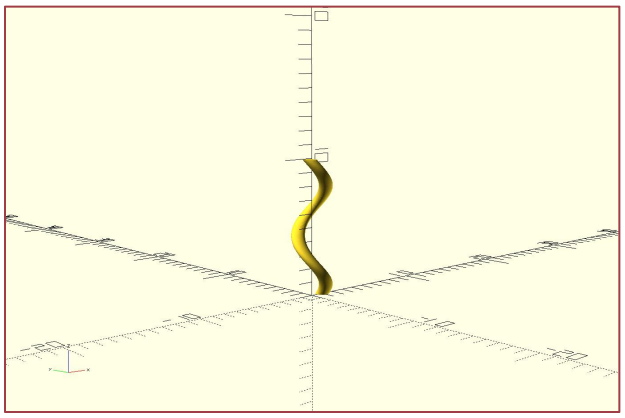




```
$fa = 1;
$fs = 0.4;

cylinder(h= 10.1999,r= 5.499,center=true);
```

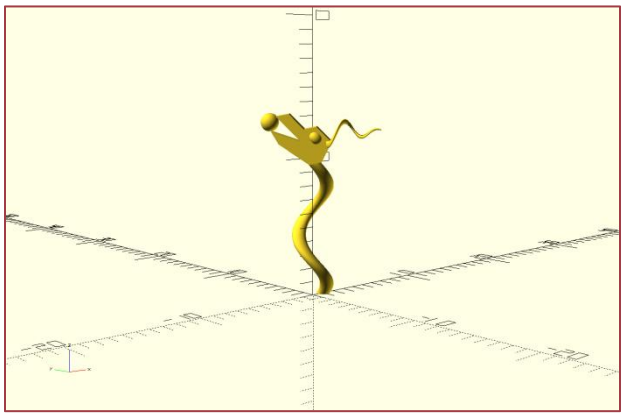
Yizhen Sun's model

1. It all starts from a simple linear_extrude...



```
/*Body*/
scale([0.5, 0.5, 0.5]){
        linear_extrude(height = 20, twist = 500)
        translate([2, 0, 0])
        circle(1);
}
}
```

2. It reminds me of some kind of dragons, so why not? Just some transformation of simple geometry. Only one thing to say is that I use linear_extrude, moving a polygon to create the head.

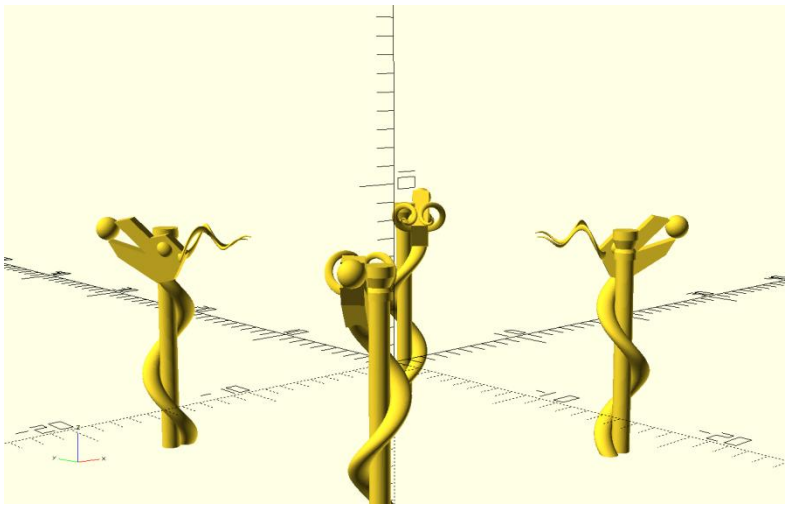


```
/*Dragon*/

/*Head*/
translate([-2.1, 1.4, 13]){
        scale(v = [0.5, 0.5, 0.5]){
                rotate([90, 45, -45]){
```

```
                linear_extrude(2.1){
                    polygon(points = [
                        [0, 0], [1, 1.5], [4, 1.5], [5, 3], [8, 3], [9, 0]
                        ]
                    );
                }

                linear_extrude(2.1){
                    polygon(points = [
                    [1, -2], [3, -2.5], [8, -1.5], [9, 0], [5, 0]
                        ]
                    );
                }
            }
        }
}

/*Eye*/
translate([-0.8, -1.2, 11.5]){
    rotate([45, 0, 45]){
        resize(newsize=[0.8, 1, 0.8])
        sphere(r=1);

        translate([0.8, 0, 0])
        resize(newsize=[0.8, 1, 0.8])
        sphere(r=1);
    }
}

/*Horn*/
translate([-0.6, -1.8, 12]){
    rotate(a = 2, v = [-0.7071,0.7071,0]){
    rotate(a = 90, v = [0.7071,0.7071,0]){
        scale([0.05, 0.05, 0.005]){
            linear_extrude(height = 800, scale = 0.1, twist = 600)
            translate([20, 1, 0])
            circle(5);
        }
    }
}
}

translate([-0.6, -1.8, 12]){
    rotate(a = 2, v = [-0.7071,0.7071,0]){
```

```
        rotate(a = 90, v = [0.7071,0.7071,0]){
            mirror([1, 1, 0]){
            translate([-1, -1, 0]){
            scale([0.05, 0.05, 0.005]){
                linear_extrude(height = 800, scale = 0.1, twist = 600)
                translate([20, 1, 0])
                circle(5);
            }
            }
            }
        }
        }
}

/*Ball*/
translate([-2.8, 1.37, 12.57])
sphere(0.64);

/*Body*/
scale([0.5, 0.5, 0.5]){
        linear_extrude(height = 20, twist = 500)
        translate([2, 0, 0])
        circle(1);
}
}
```

3. Create a pillar with some cylinder and mirror the whole object, twice.



```
/*Dragon with column*/

        /*1*/
```

```
translate([-9, 9, -5]){

/*column*/
cylinder(12, 0.5, 0.5);

/*Chapiter*/
translate([0, 0, 12])
cylinder(0.5, 0.7, 0.7);

translate([0, 0, 11.5])
cylinder(0.5, 0.5, 0.7);

translate([0, 0, 11.3])
cylinder(0.2, 0.6, 0.5);

translate([0, 0, 10.9])
cylinder(0.4, 0.6, 0.6);

/*Dragon*/

/*Head*/
translate([-2.1, 1.4, 13]){
	scale(v = [0.5, 0.5, 0.5]){
		rotate([90, 45, -45]){
			linear_extrude(2.1){
				polygon(points = [
					[0, 0], [1, 1.5], [4, 1.5], [5, 3], [8, 3], [9, 0]
					]
				);
			}

			linear_extrude(2.1){
				polygon(points = [
				[1, -2], [3, -2.5], [8, -1.5], [9, 0], [5, 0]
					]
				);
			}
		}
	}
}

/*Eye*/
translate([-0.8, -1.2, 11.5]){
	rotate([45, 0, 45]){
```

```
            resize(newsize=[0.8, 1, 0.8])
            sphere(r=1);

            translate([0.8, 0, 0])
            resize(newsize=[0.8, 1, 0.8])
            sphere(r=1);
      }
}

/*Horn*/
translate([-0.6, -1.8, 12]){
      rotate(a = 2, v = [-0.7071,0.7071,0]){
      rotate(a = 90, v = [0.7071,0.7071,0]){
            scale([0.05, 0.05, 0.005]){
                  linear_extrude(height = 800, scale = 0.1, twist = 600)
                  translate([20, 1, 0])
                  circle(5);
            }
      }
}
}

translate([-0.6, -1.8, 12]){
      rotate(a = 2, v = [-0.7071,0.7071,0]){
      rotate(a = 90, v = [0.7071,0.7071,0]){
            mirror([1, 1, 0]){
            translate([-1, -1, 0]){
            scale([0.05, 0.05, 0.005]){
                  linear_extrude(height = 800, scale = 0.1, twist = 600)
                  translate([20, 1, 0])
                  circle(5);
            }
            }
            }
      }
      }
}

/*Ball*/
translate([-2.8, 1.37, 12.57])
sphere(0.64);

/*Body*/
scale([0.5, 0.5, 0.5]){
```

```
        linear_extrude(height = 20, twist = 500)
        translate([2, 0, 0])
        circle(1);
}
}

/*2*/
mirror([0, 1, 0]){
translate([-9, 9, -5]){

/*column*/
cylinder(12, 0.5, 0.5);

/*Chapiter*/
translate([0, 0, 12])
cylinder(0.5, 0.7, 0.7);

translate([0, 0, 11.5])
cylinder(0.5, 0.5, 0.7);

translate([0, 0, 11.3])
cylinder(0.2, 0.6, 0.5);

translate([0, 0, 10.9])
cylinder(0.4, 0.6, 0.6);

/*Dragon*/

/*Head*/
translate([-2.1, 1.4, 13]){
        scale(v = [0.5, 0.5, 0.5]){
                rotate([90, 45, -45]){
                        linear_extrude(2.1){
                                polygon(points = [
                                        [0, 0], [1, 1.5], [4, 1.5], [5, 3], [8, 3], [9, 0]
                                        ]
                                );
                        }

                        linear_extrude(2.1){
                                polygon(points = [
                                [1, -2], [3, -2.5], [8, -1.5], [9, 0], [5, 0]
                                        ]
                                );
```

```
                }
            }
        }
}

/*Eye*/
translate([-0.8, -1.2, 11.5]){
        rotate([45, 0, 45]){
            resize(newsize=[0.8, 1, 0.8])
            sphere(r=1);

            translate([0.8, 0, 0])
            resize(newsize=[0.8, 1, 0.8])
            sphere(r=1);
        }
}

/*Horn*/
translate([-0.6, -1.8, 12]){
        rotate(a = 2, v = [-0.7071,0.7071,0]){
        rotate(a = 90, v = [0.7071,0.7071,0]){
            scale([0.05, 0.05, 0.005]){
                linear_extrude(height = 800, scale = 0.1, twist = 600)
                translate([20, 1, 0])
                circle(5);
            }
        }
}
}

translate([-0.6, -1.8, 12]){
        rotate(a = 2, v = [-0.7071,0.7071,0]){
        rotate(a = 90, v = [0.7071,0.7071,0]){
            mirror([1, 1, 0]){
            translate([-1, -1, 0]){
            scale([0.05, 0.05, 0.005]){
                linear_extrude(height = 800, scale = 0.1, twist = 600)
                translate([20, 1, 0])
                circle(5);
            }
            }
            }
        }
        }
```

```
}

/*Ball*/
translate([-2.8, 1.37, 12.57])
sphere(0.64);

/*Body*/
scale([0.5, 0.5, 0.5]){
	linear_extrude(height = 20, twist = 500)
	translate([2, 0, 0])
	circle(1);
}
}
}

mirror([1, 0, 0]){
/*3*/
translate([-9, 9, -5]){

/*column*/
cylinder(12, 0.5, 0.5);

/*Chapiter*/
translate([0, 0, 12])
cylinder(0.5, 0.7, 0.7);

translate([0, 0, 11.5])
cylinder(0.5, 0.5, 0.7);

translate([0, 0, 11.3])
cylinder(0.2, 0.6, 0.5);

translate([0, 0, 10.9])
cylinder(0.4, 0.6, 0.6);

/*Dragon*/

/*Head*/
translate([-2.1, 1.4, 13]){
	scale(v = [0.5, 0.5, 0.5]){
		rotate([90, 45, -45]){
			linear_extrude(2.1){
				polygon(points = [
					[0, 0], [1, 1.5], [4, 1.5], [5, 3], [8, 3], [9, 0]
```

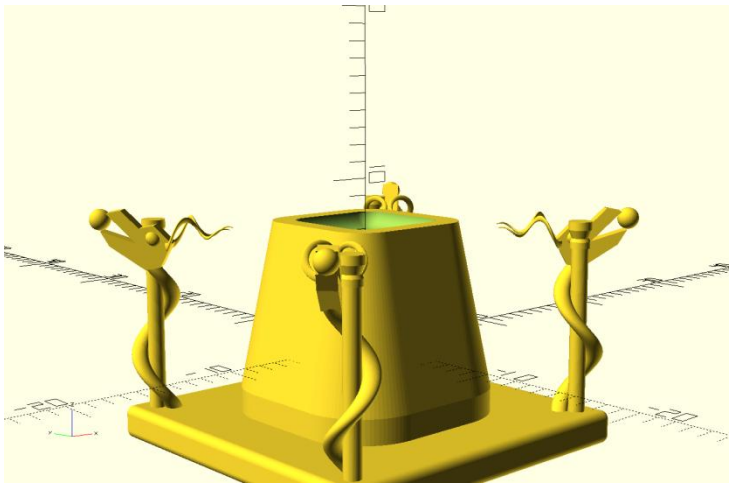
```
                        ]
                    );
                }

                linear_extrude(2.1){
                    polygon(points = [
                    [1, -2], [3, -2.5], [8, -1.5], [9, 0], [5, 0]
                        ]
                    );
                }
            }
        }
}

/*Eye*/
translate([-0.8, -1.2, 11.5]){
    rotate([45, 0, 45]){
        resize(newsize=[0.8, 1, 0.8])
        sphere(r=1);

        translate([0.8, 0, 0])
        resize(newsize=[0.8, 1, 0.8])
        sphere(r=1);
    }
}

/*Horn*/
translate([-0.6, -1.8, 12]){
    rotate(a = 2, v = [-0.7071,0.7071,0]){
    rotate(a = 90, v = [0.7071,0.7071,0]){
        scale([0.05, 0.05, 0.005]){
            linear_extrude(height = 800, scale = 0.1, twist = 600)
            translate([20, 1, 0])
            circle(5);
        }
    }
}
}

translate([-0.6, -1.8, 12]){
    rotate(a = 2, v = [-0.7071,0.7071,0]){
    rotate(a = 90, v = [0.7071,0.7071,0]){
        mirror([1, 1, 0]){
        translate([-1, -1, 0]){
```

```
            scale([0.05, 0.05, 0.005]){
                linear_extrude(height = 800, scale = 0.1, twist = 600)
                translate([20, 1, 0])
                circle(5);
            }
            }
            }
      }
      }
}

/*Ball*/
translate([-2.8, 1.37, 12.57])
sphere(0.64);

/*Body*/
scale([0.5, 0.5, 0.5]){
      linear_extrude(height = 20, twist = 500)
      translate([2, 0, 0])
      circle(1);
}
}

mirror([0, 1, 0]){
translate([-9, 9, -5]){

/*4*/
/*column*/
cylinder(12, 0.5, 0.5);

/*Chapiter*/
translate([0, 0, 12])
cylinder(0.5, 0.7, 0.7);

translate([0, 0, 11.5])
cylinder(0.5, 0.5, 0.7);

translate([0, 0, 11.3])
cylinder(0.2, 0.6, 0.5);

translate([0, 0, 10.9])
cylinder(0.4, 0.6, 0.6);

/*Dragon*/
```

```
/*Head*/
translate([-2.1, 1.4, 13]){
	scale(v = [0.5, 0.5, 0.5]){
		rotate([90, 45, -45]){
			linear_extrude(2.1){
				polygon(points = [
					[0, 0], [1, 1.5], [4, 1.5], [5, 3], [8, 3], [9, 0]
					]
				);
			}

			linear_extrude(2.1){
				polygon(points = [
				[1, -2], [3, -2.5], [8, -1.5], [9, 0], [5, 0]
					]
				);
			}
		}
	}
}

/*Eye*/
translate([-0.8, -1.2, 11.5]){
	rotate([45, 0, 45]){
		resize(newsize=[0.8, 1, 0.8])
		sphere(r=1);

		translate([0.8, 0, 0])
		resize(newsize=[0.8, 1, 0.8])
		sphere(r=1);
	}
}

/*Horn*/
translate([-0.6, -1.8, 12]){
	rotate(a = 2, v = [-0.7071,0.7071,0]){
	rotate(a = 90, v = [0.7071,0.7071,0]){
		scale([0.05, 0.05, 0.005]){
			linear_extrude(height = 800, scale = 0.1, twist = 600)
			translate([20, 1, 0])
			circle(5);
		}
	}
```

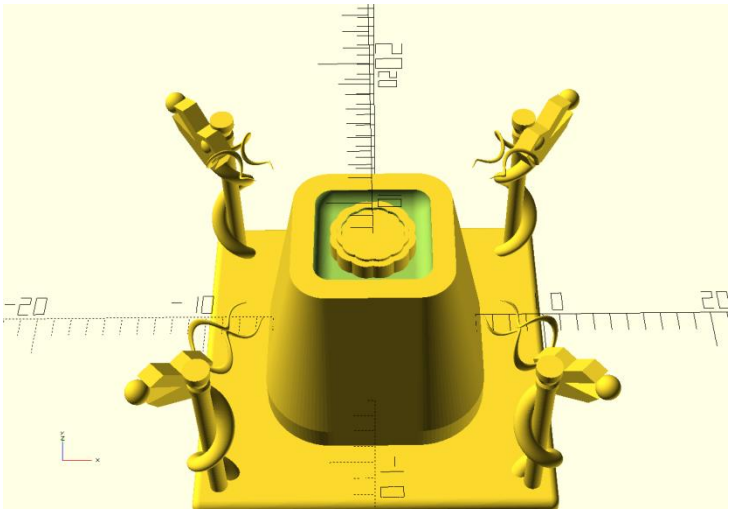
```
}
}

translate([-0.6, -1.8, 12]){
	rotate(a = 2, v = [-0.7071,0.7071,0]){
	rotate(a = 90, v = [0.7071,0.7071,0]){
		mirror([1, 1, 0]){
		translate([-1, -1, 0]){
		scale([0.05, 0.05, 0.005]){
			linear_extrude(height = 800, scale = 0.1, twist = 600)
			translate([20, 1, 0])
			circle(5);
		}
		}
		}
	}
	}
}

/*Ball*/
translate([-2.8, 1.37, 12.57])
sphere(0.64);

/*Body*/
scale([0.5, 0.5, 0.5]){
	linear_extrude(height = 20, twist = 500)
	translate([2, 0, 0])
	circle(1);
}
}
}
}
```

4. I love minkowski, don't know how exactly it works (I mean the mathematical calculation behind) but it help me made smoother geometries. Also use difference to create the hollow on the top.



```
/*True Box*/
        difference(){
            difference(){
                rotate([180, 0, 0]){
                    minkowski(){
                        cube(5, center = true);
                        cylinder(h = 10, r1 = 2, r2 = 4, center = true);
                    }
                }

            translate([0, 0, -10])
            cube([20, 20, 12], center = true);
            }

            translate([0, 0, 7.5]){
                minkowski(){
                    cube([4.5, 4.5, 0.01], center = true);
                    sphere(1);
                }
            }
        }

        translate([0, 0, -5]){
            minkowski(){
                cube([20, 20, 1], center = true);
                sphere(0.5);
            }
        }
```

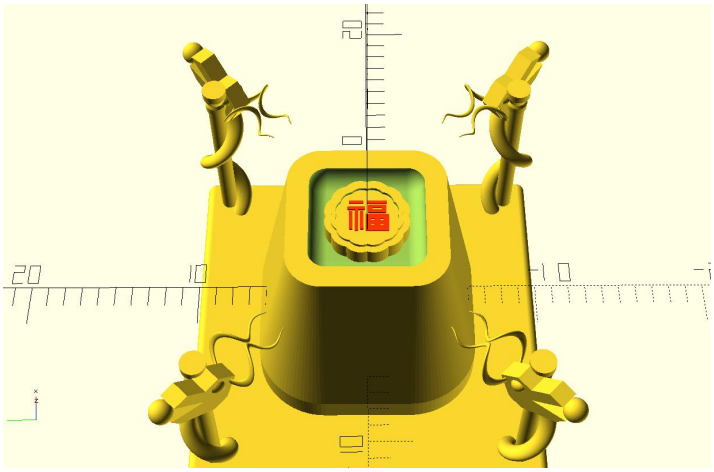
5. Use a for loop to create the ruffles. Resize it twice and difference them to get the pattern above.



```
/*Pastry*/
    translate([0, 0, 6.75]){

        /*Base*/
        cylinder(h = 1.5, r = 2, center = true);

        for(i = [0 : 30 : 360]){
            rotate([0, 0, i]){
                translate([1, 1, 0])
                cylinder(h = 1.5, r = 1, center = true);
            }
        }

        /*pattern*/
        difference(){
        resize([4, 4, 0.8]){
            translate([0, 0, 1])
            cylinder(h = 1.5, r = 2, center = true);
            for(i = [0 : 30 : 360]){
                rotate([0, 0, i]){
                    translate([1, 1, 1])
                    cylinder(h = 1.5, r = 1, center = true);
                }
            }
        }

        resize([3.5, 3.5, 0.9]){
            translate([0, 0, 1])
            cylinder(h = 1.5, r = 2, center = true);
            for(i = [0 : 30 : 360]){
                rotate([0, 0, i]){
```

```
                translate([1, 1, 1])
                cylinder(h = 1.5, r = 1, center = true);
            }
        }
    }
    }
}
```

6. I find how to create English characters easily but hey, Chinese pastry will be better with Chinese character. So I make some cylinders and form a character using them.



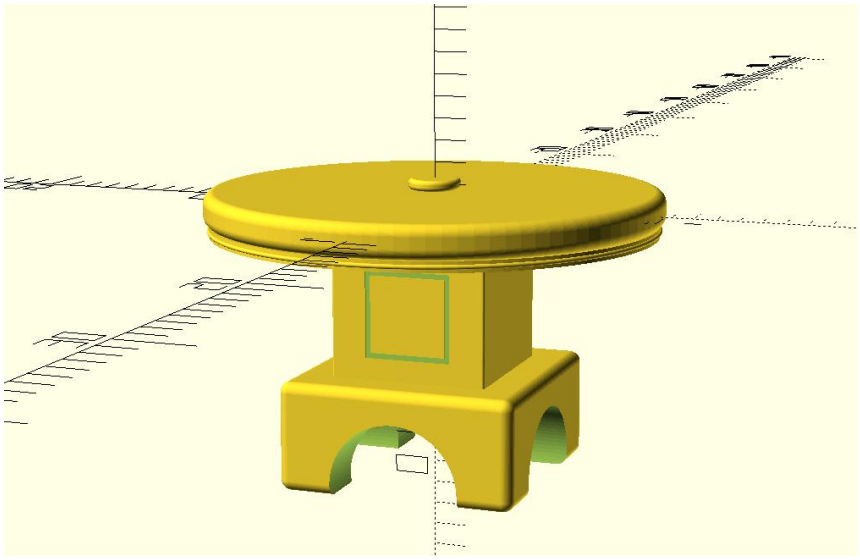
```
translate([0, 0, 6.75]){
    /*Character*/
    translate([0, 0.05, 0]){
    color( c = [252 / 255, 52 / 255, 3 / 255], alpha = 1.0 ){
        translate([0.85, 0.2, 0.75]){
            rotate([0, 0, 0]){
                minkowski(){
                    cube([0.1, 0.7, 0.05]);
                    cylinder(h = 0.05, r = 0.05, center = true);
                }
            }
        }
    }

    color( c = [252 / 255, 52 / 255, 3 / 255], alpha = 1.0 ){
        translate([0.45, 0.1, 0.75]){
            rotate([0, 0, 0]){
                minkowski(){
```

```
                cube([0.1, 0.9, 0.05]);
                cylinder(h = 0.05, r = 0.05, center = true);
            }
        }
    }
}

color( c = [252 / 255, 52 / 255, 3 / 255], alpha = 1.0 ){
    translate([0.45, 0.5, 0.75]){
        rotate([0, 0, 90]){
            minkowski(){
                cube([0.1, 1.4, 0.05]);
                cylinder(h = 0.05, r = 0.05, center = true);
            }
        }
    }
}

color( c = [252 / 255, 52 / 255, 3 / 255], alpha = 1.0 ){
    translate([0.25, 0.8, 0.75]){
        rotate([0, 0, 90]){
            minkowski(){
                cube([0.1, 1.1, 0.05]);
                cylinder(h = 0.05, r = 0.05, center = true);
            }
        }
    }
}

color( c = [252 / 255, 52 / 255, 3 / 255], alpha = 1.0 ){
    translate([0.25, 0.2, 0.75]){
        rotate([0, 0, 90]){
            minkowski(){
                cube([0.1, 1.1, 0.05]);
                cylinder(h = 0.05, r = 0.05, center = true);
            }
        }
    }
}

color( c = [252 / 255, 52 / 255, 3 / 255], alpha = 1.0 ){
    translate([0.85, -1.1, 0.75]){
        rotate([0, 0, 0]){
            minkowski(){
```

```
                cube([0.1, 1, 0.05]);
                cylinder(h = 0.05, r = 0.05, center = true);
            }
        }
    }
}

color( c = [252 / 255, 52 / 255, 3 / 255], alpha = 1.0 ){
    translate([0.25, -1.05, 0.75]){
        rotate([0, 0, 0]){
            difference(){
                minkowski(){
                    cube([0.4, 0.9, 0.05]);
                    cylinder(h = 0.05, r = 0.05, center = true);
                }

                translate([0.165, 0.195, -0.1])
                cube([0.1, 0.5, 1]);
            }
        }
    }
}

color( c = [252 / 255, 52 / 255, 3 / 255], alpha = 1.0 ){
    translate([-0.95, -1.2, 0.75]){
        rotate([0, 0, 0]){
            difference(){
                minkowski(){
                    cube([1, 1.2, 0.05]);
                    cylinder(h = 0.05, r = 0.05, center = true);
                }

                translate([0.32, 0.26, -0.1])
                cube([0.12, 0.17, 1]);

                translate([0.62, 0.26, -0.1])
                cube([0.12, 0.17, 1]);

                translate([0.32, 0.75, -0.1])
                cube([0.12, 0.17, 1]);

                translate([0.62, 0.75, -0.1])
                cube([0.12, 0.17, 1]);
            }
```

```
                }
            }
        }
    }
    }
```

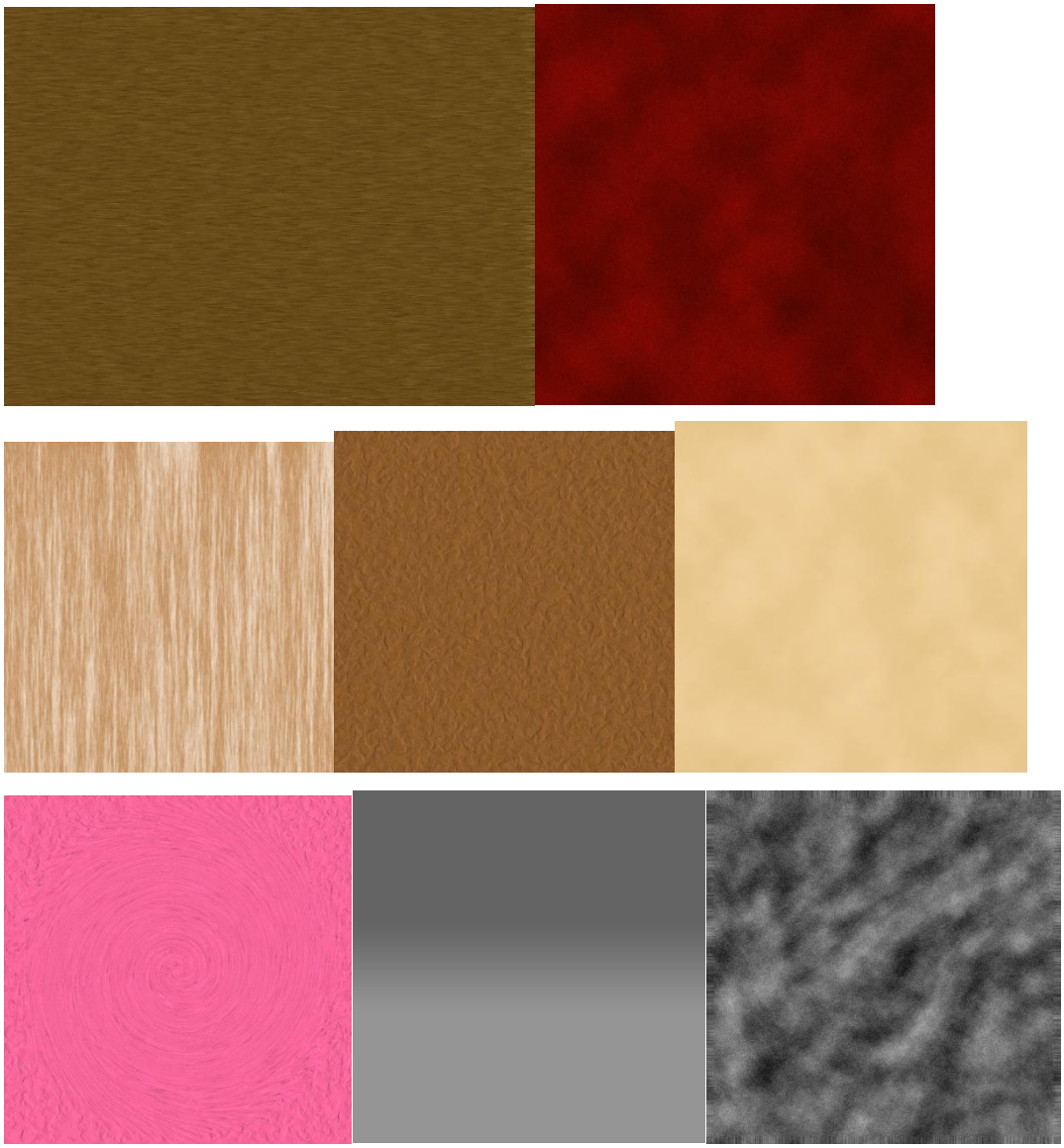
7. Create a simple table. Again I love minkowski.



```
$fn = 100;
/*Table*/
    /*Top*/
    translate([0, 0, 0]){
        minkowski(){
            cylinder(h = 0.5, r = 10, center = true);
            sphere(r = 0.5);
        }
    }

    translate([0, 0, -1])
    cylinder(h = 0.5, r = 10.25, center = true);

    translate([0, 0, -1]){
        rotate_extrude(angle = 360, convexity = 2){
            translate([10.25, 0, 0])
            circle(r = 0.1);
        }
    }

    translate([0, 0, 1]){
        minkowski(){
```

```
            cylinder(h = 0.1, r = 1, center = true);
            sphere(r = 0.2);
        }
    }

    /*Bottom*/
    translate([0, 0, -3.5]){
        difference(){
            minkowski(){
                cube([7, 7, 8], center = true);
                sphere(0.1);
            }
        translate([0, 5.5, -1])
        cube([4, 4, 4], center = true);
        }

        translate([0, 1.625, -1])
        cube([3.5, 4, 3.5], center = true);

        translate([0, 0, -6.5]){
            difference(){
                minkowski(){
                    cube([10, 10, 4], center = true);
                    sphere(0.5);
                }

                translate([0, 0, -2])
                rotate([0, 90, 0])
                cylinder(h = 11, r = 3, center = true);

                translate([0, 0, -2])
                rotate([0, 90, 90])
                cylinder(h = 11, r = 3, center = true);
            }
        }
    }
```

8. Make some textures with Photoshop.

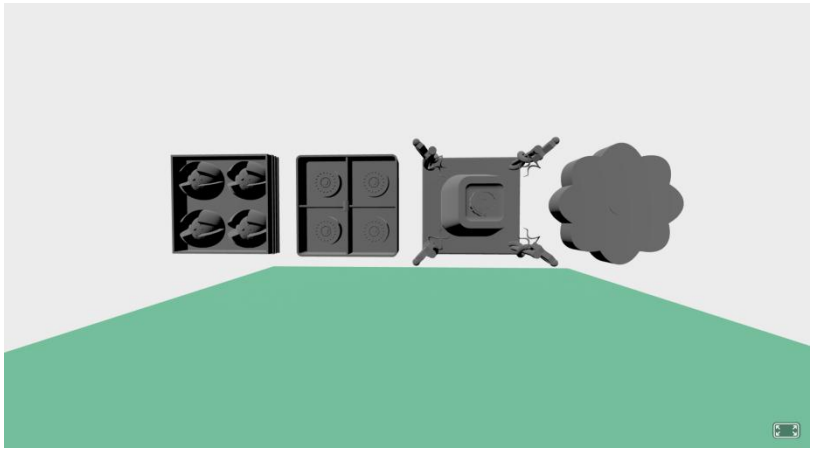


Phase 2:

In order to facilitate the next steps, Yizhen Sun put all our models together and arranged them in a more rational and beautiful way.

Glitch:

Zhengxuan Li Import the four models into Glitch to get the VR effect (it's really fun).

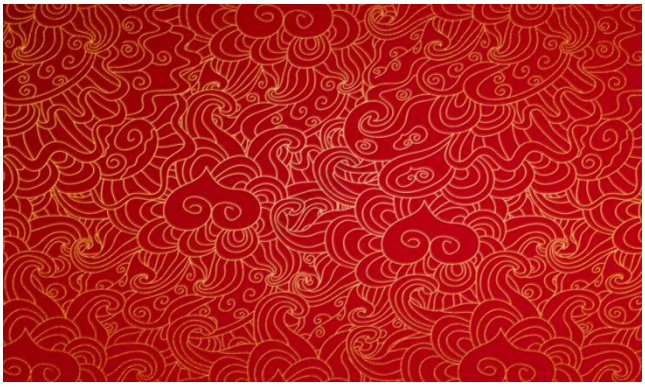


```
<html>
  <head>
    <script src="https://aframe.io/releases/1.7.0/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>

      <a-assets>
        <a-asset-item id="model1" src="https://cdn.glitch.global/2cf984f6-0c15-4134-a84f-6da2ec242b88/test2.gltf?v=1742450287862"></a-asset-item>
        <a-asset-item id="model2" src="https://cdn.glitch.global/2cf984f6-0c15-4134-a84f-6da2ec242b88/sweetheart%20pastry02.gltf?v=1742452208303"></a-asset-item>
        <a-asset-item id="model3" src="https://cdn.glitch.global/2cf984f6-0c15-4134-a84f-6da2ec242b88/pastry.gltf?v=1742452360268"></a-asset-item>
        <a-asset-item id="model4" src="https://cdn.glitch.me/2cf984f6-0c15-4134-a84f-6da2ec242b88/Pastry1.gltf?v=1742475623895"></a-asset-item>
      </a-assets>


      <a-entity gltf-model="#model1" position="-5 2 -7" scale="0.05 0.05 0.05"></a-entity>
      <a-entity gltf-model="#model2" position="-3 0.7 -7" scale="0.025 0.025 0.025"></a-entity>
      <a-entity gltf-model="#model3" position="5 2 -7" scale="0.05 0.05 0.05"></a-entity>
      <a-entity gltf-model="#model4" position="1.5 2 -7" scale="0.13 0.13 0.13"></a-entity>


      <a-plane position="0 0 -4" rotation="-90 0 0" width="10" height="10" color="#7BC8A4"></a-plane>
      <a-sky color="#ECECEC"></a-sky>
    </a-scene>
  </body>
</html>

```

Rendering:

It is mainly handled by Yizhen Sun.

Zhengxuan Li adds a background image to a web page.



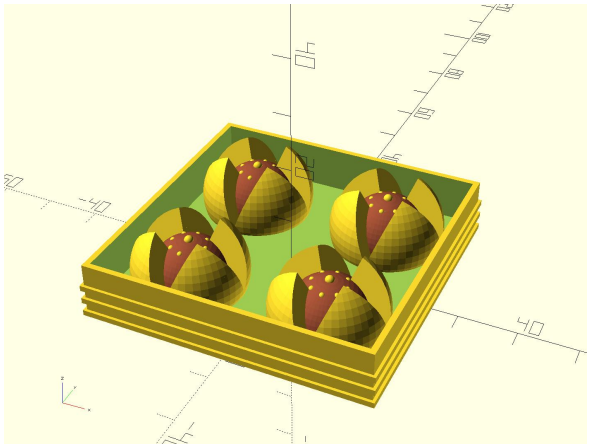
```css
#background {
    position: fixed;
    top: 0; left: 0; width: 100%; height: 100%;
    background-image: url('img/2.jpg');
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;
    background-attachment: fixed;
    z-index: -1;
    opacity: 0.8;
    filter: brightness(0.5) saturate(1);
}
```

Animation：

Contribute description:

Zhengxuan Li:

1.Create a model of Water Lily Crisp.



2. Import the model into three.js.

3. Create the webpage background image.

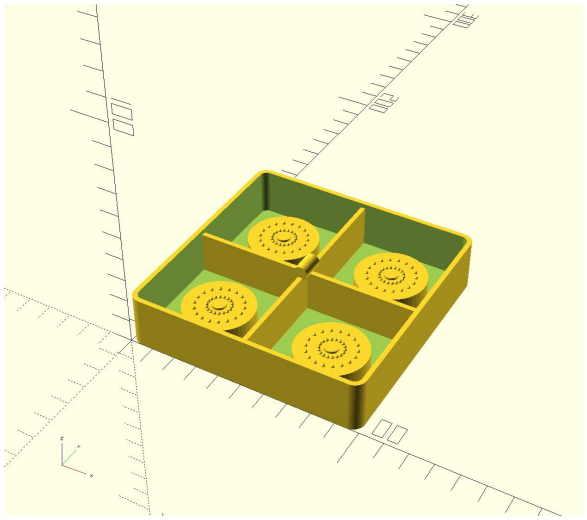
4. create model animation effect and GUI panel (may not be used in the end).

5. import the model into VR-browser: Glitch.

6. Layout the report PPT.

Weiyu Ouyang:

1.Create a model of sweetheart pastry



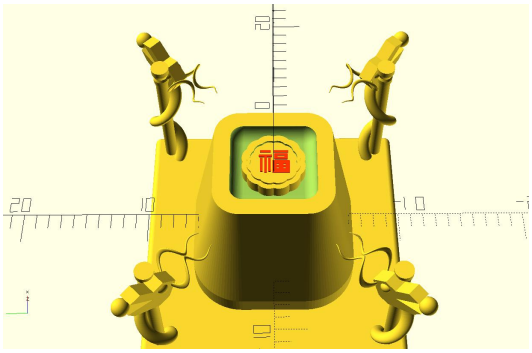
2.Be responsible for writing the final report

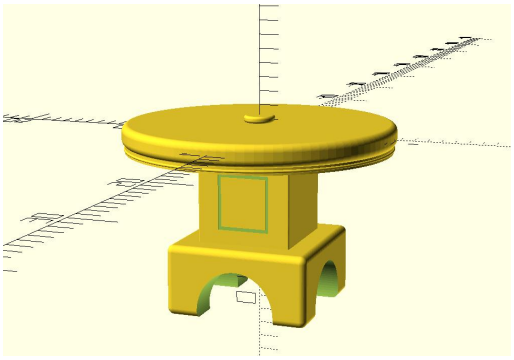
3.Create a theme template for PowerPoint presentations.

4.Participate in the production of the final PPT

Yizhen Sun:

1. 2 models, mooncake with box and the table in "\Final\Model\SYZ"





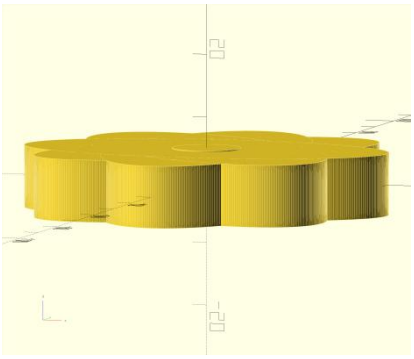
2. Every ".png" textures in "\Final\Texture\SYZ"

3. Whole index.html and index.js in "\Final"

4. PPT and Word report for the above

Youyou Wu:

1.Create a model of peach blossom pastry



2.Designed the color card matching for others modeling work (won't be used all as the final version)

3.Participate in the production of the final PPT

4.Be responsible for the writing of the final presentation

5.Discuss the final presentation with the associate supervisor

Code:

HTML

Yizhen Sun Write a simple HTML, adding a import map for Three.js library and Three.js example modules. Also import GSAP and load index.js as a JavaScript module.

```
<!DOCTYPE html>
<html lang = "en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>demo</title>
        <style>
            body{
                margin: 0;
```

```
            }
        </style>
        <script type="importmap">
            {
                "imports": {
                    "three": "https://cdn.jsdelivr.net/npm/three@0.174.0/build/three.module.js",
                    "jsm/": "https://cdn.jsdelivr.net/npm/three@0.174.0/examples/jsm/"
                }
            }
        </script>
        <script src="https://cdn.jsdelivr.net/npm/gsap@3.11.1/dist/gsap.min.js"></script>
    </head>
    <body>
        <script type="module" src="index.js"></script>
    </body>
</html>
import * as THREE from "three";
import { STLLoader } from "jsm/loaders/STLLoader.js";
import { OrbitControls } from "jsm/controls/OrbitControls.js";

const scene = new THREE.Scene();
```

```
/*Renderer*/
const w = window.innerWidth;
const h = window.innerHeight;
const renderer = new THREE.WebGLRenderer({ antialias: true });
renderer.setSize(w, h);
document.body.appendChild(renderer.domElement);
```

```
/*Camera*/
const fov = 75;
const aspect = w / h;
const near = 0.1;
const far = 100;
const camera = new THREE.PerspectiveCamera(fov, aspect, near, far);
camera.position.set(0, 0, 5);
```

```
/*Light*/
const ambientLight = new THREE.AmbientLight(0xffffff, 0.4);
scene.add(ambientLight);
const dirlight = new THREE.DirectionalLight(0xffffff, 1);
dirlight.position.set(0, 20, 0);
scene.add(dirlight);
/*Texture*/
```

```
const textureloader = new THREE.TextureLoader();
    const Rind = textureloader.load('Texture/LZX/Rind.jpg');
    const Kernel = textureloader.load('Texture/LZX/Kernel.jpg');
    const Box2 = textureloader.load('Texture/SYZ/Box2.png');
    const Character = textureloader.load('Texture/SYZ/Character.png');
    const Pastry1 = textureloader.load('Texture/SYZ/Pastry1.png');
    const Box1 = textureloader.load('Texture/SYZ/Stone.png');
    const Pastry3 = textureloader.load('Texture/SYZ/Pastry3.png');
    const Box3 = textureloader.load('Texture/SYZ/Steel.png');
    const Pastry4 = textureloader.load('Texture/SYZ/Pastry4.png');
    const Wood = textureloader.load('Texture/SYZ/Wood.png');

/*Background*/
textureloader.load('Texture/LZX/Background.png', function(texture){
    scene.background = texture;
})
```

```
/*UV*/
function CylinderUV(geometry) {
    geometry.computeBoundingBox();
    const center = geometry.boundingBox.getCenter(new THREE.Vector3());
    const size = geometry.boundingBox.getSize(new THREE.Vector3());

    geometry.setAttribute('uv', new THREE.BufferAttribute(new Float32Array(geometry.attributes.position.count
* 2), 2));

    for (let i = 0; i < geometry.attributes.position.count; i++) {
      const vertex = new THREE.Vector3(
        geometry.attributes.position.getX(i),
        geometry.attributes.position.getY(i),
        geometry.attributes.position.getZ(i)
      );

      vertex.sub(center).divide(size);

      const u = 0.5 + Math.atan2(vertex.z, vertex.x) / (2 * Math.PI);
      const v = (vertex.y + 0.5);

      geometry.attributes.uv.setXY(i, u, v);
    }
}
```

```
function SphereUV(geometry){
    geometry.computeBoundingBox();
```

```
    const bboxSize = geometry.boundingBox.getSize(new THREE.Vector3());

    geometry.setAttribute('uv', new THREE.BufferAttribute(new Float32Array(geometry.attributes.position.count
* 2), 2));

    for (let i = 0; i < geometry.attributes.position.count; i++) {
        const x = geometry.attributes.position.getX(i);
        const y = geometry.attributes.position.getY(i);
        const z = geometry.attributes.position.getZ(i);

        const absX = Math.abs(x / bboxSize.x);
        const absY = Math.abs(y / bboxSize.y);
        const absZ = Math.abs(z / bboxSize.z);

        if (absX >= absY && absX >= absZ) {
        geometry.attributes.uv.setXY(i, (z - geometry.boundingBox.min.z) / bboxSize.z,
                                        (y - geometry.boundingBox.min.y) / bboxSize.y);
        } else if (absY >= absX && absY >= absZ) {
        geometry.attributes.uv.setXY(i, (x - geometry.boundingBox.min.x) / bboxSize.x,
                                        (z - geometry.boundingBox.min.z) / bboxSize.z);
        } else {
        geometry.attributes.uv.setXY(i, (x - geometry.boundingBox.min.x) / bboxSize.x,
                                        (y - geometry.boundingBox.min.y) / bboxSize.y);
        }
    }
}
/*Models*/
const loader = new STLLoader();
const PastryS = new THREE.Group();
const PastryL = new THREE.Group();
const PastryO = new THREE.Group();
const PastryW = new THREE.Group();
const Table = new THREE.Group();
    /*SYZ*/
        /*Character*/
        loader.load('Model/SYZ/Character.stl', function(geometry){
            CylinderUV(geometry);

            const material = new THREE.MeshStandardMaterial({
                map: Character,
                roughness: 0.5,
                metalness: 0.7,
            });
            const mesh = new THREE.Mesh(geometry, material);
```

```
        mesh.scale.set(0.1, 0.1, 0.1);

        mesh.rotation.x -= 1.5;
        mesh.rotation.z += 0.785;

        PastryS.add(mesh);
    });

    /*Pastry1*/
    loader.load('Model/SYZ/Pastry1.stl', function(geometry){
        CylinderUV(geometry);

        const material = new THREE.MeshStandardMaterial({
            map: Pastry1,
            roughness: 0.8,
            metalness: 0.1,
        });
        const mesh = new THREE.Mesh(geometry, material);
        mesh.scale.set(0.1, 0.1, 0.1);

        mesh.rotation.x -= 1.5;

        PastryS.add(mesh);
    });

    /*Box1*/
    loader.load('Model/SYZ/Box1.stl', function(geometry){
        SphereUV(geometry);

        const material = new THREE.MeshStandardMaterial({
            map: Box1,
            roughness: 0.8,
            metalness: 0.6,
        });
        const mesh = new THREE.Mesh(geometry, material);
        mesh.scale.set(0.1, 0.1, 0.1);

        mesh.rotation.x -= 1.5;
        mesh.rotation.z -= 2.355;

        PastryS.add(mesh);
        PastryS.position.set(-4, -1.1, -1);
        scene.add(PastryS);
    });
```

```
    /*Table*/
    loader.load('Model/SYZ/Table.stl', function(geometry){
        CylinderUV(geometry);
```

```
        const material = new THREE.MeshStandardMaterial({
            map: Wood,
            roughness: 0.8,
            metalness: 0.5,
        });
        const mesh = new THREE.Mesh(geometry, material);
        mesh.scale.set(0.8, 0.8, 0.8);
```

```
        mesh.rotation.x -= 1.5;
        mesh.rotation.z -= 3.14;
```

```
        Table.add(mesh);
        Table.position.set(0, -2, -5);
        scene.add(Table);
    });
```

```
/*LZX*/
    /*Rind*/
    loader.load('Model/LZX/Rind.stl', function(geometry){
        SphereUV(geometry);
```

```
        const material = new THREE.MeshStandardMaterial({
            map: Rind,
            roughness: 0.8,
            metalness: 0.1,
        });
        const mesh = new THREE.Mesh(geometry, material);
        mesh.scale.set(0.04, 0.04, 0.04);
```

```
        mesh.rotation.x -= 1.5;
        mesh.rotation.z += 0.785;
```

```
        PastryL.add(mesh);
    });
```

```
    /*Kernel*/
    loader.load('Model/LZX/Kernel.stl', function(geometry){
        SphereUV(geometry);
```

```
            const material = new THREE.MeshStandardMaterial({
                map: Kernel,
                roughness: 0.8,
                metalness: 0.1,
            });
            const mesh = new THREE.Mesh(geometry, material);
            mesh.scale.set(0.04, 0.04, 0.04);
```

```
            mesh.rotation.x -= 1.5;
            mesh.rotation.z += 0.785;
```

```
            PastryL.add(mesh);
        });
```

```
        /*Box2*/
        loader.load('Model/LZX/Box2.stl', function(geometry){
            CylinderUV(geometry);
```

```
            const material = new THREE.MeshStandardMaterial({
                map: Box2,
                roughness: 0.5,
                metalness: 0.3,
            });
            const mesh = new THREE.Mesh(geometry, material);
            mesh.scale.set(0.04, 0.04, 0.04);
```

```
            mesh.rotation.x -= 1.5;
            mesh.rotation.z += 0.785;
```

```
            PastryL.add(mesh);
            PastryL.position.set(4, -1.7, -1);
            scene.add(PastryL);
        });


    /*OYWY*/
        /*Pastry3*/
        loader.load('Model/OYWY/Pastry3.stl', function(geometry){
            CylinderUV(geometry);
```

```
            const material = new THREE.MeshStandardMaterial({
                map: Pastry3,
                roughness: 0.8,
                metalness: 0.1,
            });
```

```
        const mesh = new THREE.Mesh(geometry, material);
        mesh.scale.set(0.02, 0.02, 0.02);
```

```
        mesh.rotation.x -= 1.5;
        mesh.rotation.z += 0.785;
```

```
        PastryO.add(mesh);
    });
```

```
    /*Box2*/
    loader.load('Model/OYWY/Box3.stl', function(geometry){
        CylinderUV(geometry);
```

```
        const material = new THREE.MeshStandardMaterial({
            map: Box3,
            roughness: 0.3,
            metalness: 0.8,
        });
        const mesh = new THREE.Mesh(geometry, material);
        mesh.scale.set(0.02, 0.02, 0.02);
```

```
        mesh.rotation.x -= 1.5;
        mesh.rotation.z += 0.785;
```

```
        PastryO.add(mesh);
        PastryO.position.set(4, -1.2, -7.5);
        scene.add(PastryO);
    });


/*WYY*/
    /*Pastry4*/
    loader.load('Model/WYY/Pastry4.stl', function(geometry){
        CylinderUV(geometry);
```

```
        const material = new THREE.MeshStandardMaterial({
            map: Pastry4,
            roughness: 0.7,
            metalness: 0.2,
        });
        const mesh = new THREE.Mesh(geometry, material);
        mesh.scale.set(0.04, 0.04, 0.04);
```

```
        mesh.rotation.x -= 1.5;
```

```
            PastryW.add(mesh);
            PastryW.position.set(-4, -1, -9);
            scene.add(PastryW);
        });
/*Control*/
const controls = new OrbitControls(camera, renderer.domElement);
controls.enableDamping = true;
controls.dampingFactor = 0.25;
controls.screenSpacePanning = true;

/*Choose Pastry*/
const raycaster = new THREE.Raycaster();
const mouse = new THREE.Vector2();
const targetPosition = new THREE.Vector3(0, 0, 0);
```

```
window.addEventListener('click', onMouseClick, false);
window.addEventListener('keydown', function(event){
    if(event.key ==='Escape'){
        resetScene();
        window.addEventListener('click', onMouseClick, false);
    }
})
```

```
/*Rotate*/
let isRotating = false;
let rotationTween = null;
```

```
window.addEventListener('keydown', function(event){
    if(event.key === ' '){
        if (!isRotating) {
            rotateScene();
        } else {
            stopRotation();
        }
    }
});
```

```
function rotateScene() {
    isRotating = true;
    rotationTween = gsap.to(scene.rotation, {
        y: "+=" + Math.PI * 20,
        duration: 5,
        repeat: -1,
        ease: "linear"
```

```
    });
}
```

```
function stopRotation() {
    isRotating = false;
    if (rotationTween) {
        rotationTween.kill();
        rotationTween = null;
    }
}
```

```
function onMouseClick(event){
    mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
    mouse.y = -(event.clientY / window.innerHeight) * 2 + 1;
    raycaster.setFromCamera(mouse, camera);
```

```
    const intersects = raycaster.intersectObjects(scene.children, true);
```

```
    if(intersects.length > 0){
        const selectedObject = intersects[0].object;
        const parentGroup = findParentGroup(selectedObject);
```

```
        if(parentGroup){
            focusOnGroup(parentGroup);
            window.removeEventListener('click', onMouseClick);
        }
    }
}
```

```
function findParentGroup(object){
    let current = object;
    while(current.parent){
        if(current.parent instanceof THREE.Group){
            return current.parent;
        }
        current = current.parent;
    }
}
```

```
function focusOnGroup(group){
    if (!group) return;
```

```
    [PastryS, PastryL, PastryO, PastryW, Table].forEach(g =>{
        g.visible = (g === group);
```

```
    });

    group.getWorldPosition(targetPosition);

    controls.enabled = false;
    gsap.to(camera.position, {
        x: targetPosition.x+5,
        y: targetPosition.y+5,
        z: targetPosition.z+5,
        duration: 1,
        ease: "power2.out",
        onUpdate: () => {
            controls.target.copy(targetPosition);
            controls.update();
        },
        onComplete: () => {
            controls.enabled = true;
        }
    });
}

function resetScene(){
    [PastryS, PastryL, PastryO, PastryW, Table].forEach(model =>{
        model.visible = true;
    });

    const newTargetPosition = new THREE.Vector3(0, 0, 0);

    controls.enabled = false;
    gsap.to(camera.position, {
        x: 0,
        y: 0,
        z: 5,
        duration: 1,
        ease: "power2.out",
        onUpdate: () => {
            controls.target.copy(new THREE.Vector3(0, 0, 0));
            controls.update();
        },
        onComplete: () => {
            controls.enabled = true;
        }
    });
}
```

```
/*Animation*/
function animate(){
    requestAnimationFrame(animate);
    renderer.render(scene, camera);
    controls.update();
}
```

```
animate();
```

Experimental results:

We successfully created 3DCG content of traditional Chinese dim sum using OpenSCAD and Three.js

Summary and reflection:

While the project achieved its main goal, there are still parts that could be improved related to accurately capturing the details of the Chinese Dim Sum and converting it into a digital format. Future iterations of the project could explore other features to enhance user engagement and educational value, call for the preservation of traditional Chinese culture, and export to the outside world.