Digital Manufacturing

Project 2: Lampshade Lattice

Author:

Yufan Wang

Date:

Feb/14th/2021

Steps

The initial design was a hallow sphere which is shown below, however, it was so time consuming and expensive to print. It takes up to 1 day to print the just fit size for tealight. The main cause of this is too much overhanging in this design. The support structure significantly extends the printing time and the amount of the filament.

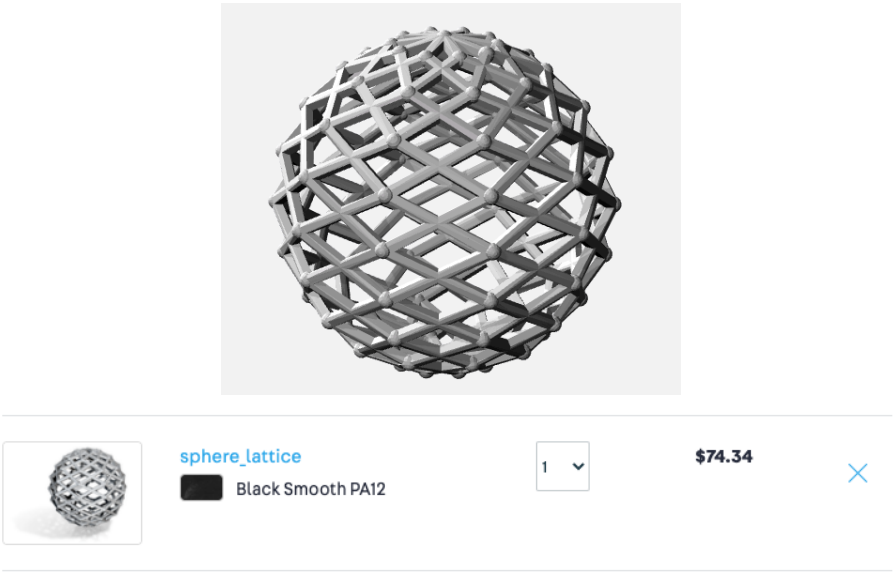


Figure 1. The initial design of the lampshade

Then I choose to design something compact and easy to print. Instead of fully auto generated design with one single pattern, I decided to make one which only its pattern is recursively generated, the other dimensions and shapes of the lampshade can be defined by the user or designer.

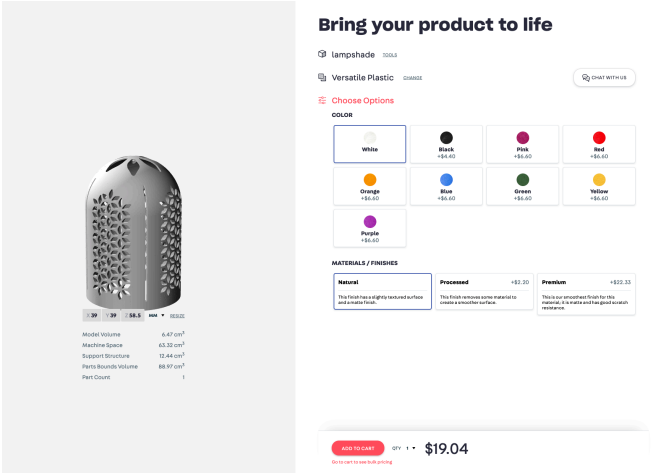


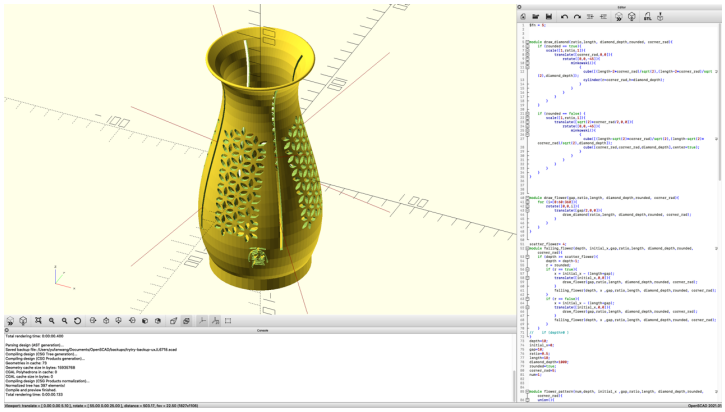
Figure 2. A new design of lampshade perfectly fit on tealight.

The new design turned out pretty good. The size and shades generated was beautiful. The images are shown following:

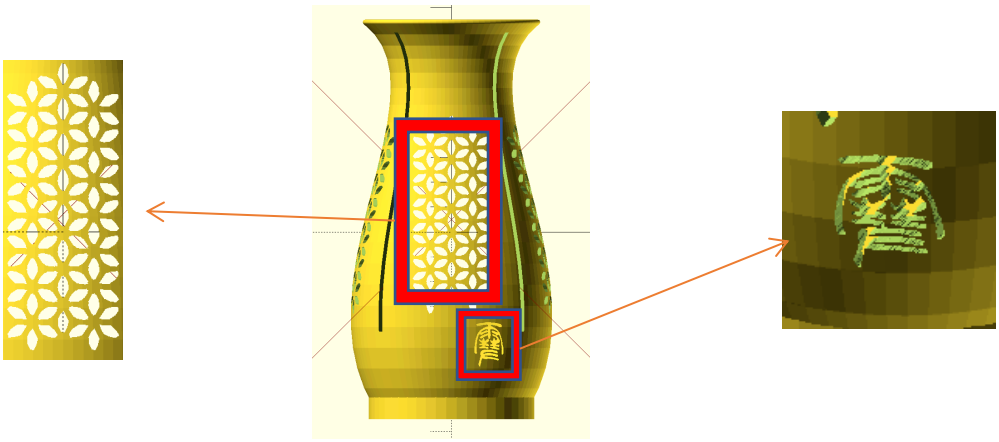
Figure 3. Lampshade printed with Hatchbox black filament.

Figure 4. Lampshade printed with Sunlu transparent filament.

After those prints are finished, I still had plenty of time to play with the modal. So I make a new vase-like lampshade. It was supposed to look fancier than the previous one, but, sometimes, the reality isn't what I think or feel. The new design is posted below.



(a)



(b)

Figure 5. (a)Vase shape tealight lampshade. (b) Design details.

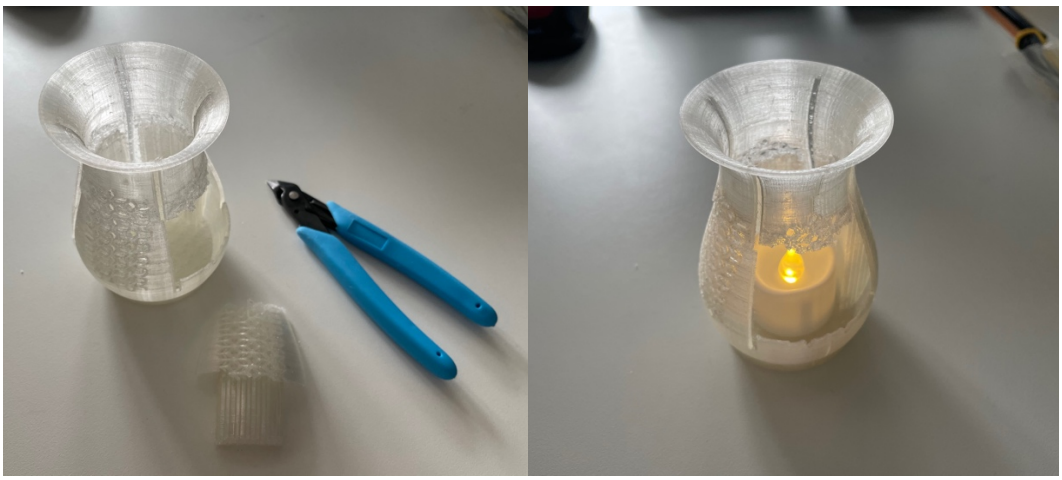


Figure 6. Image of the new vase shape print.

The support of this print was too stiff and tricky to remove, and I broke the print when removing those supports. But it has already been close to the deadline, so I decide to just go with the small lampshade design. But I did make some other attempts to make something different, like the cube pattern shown below.

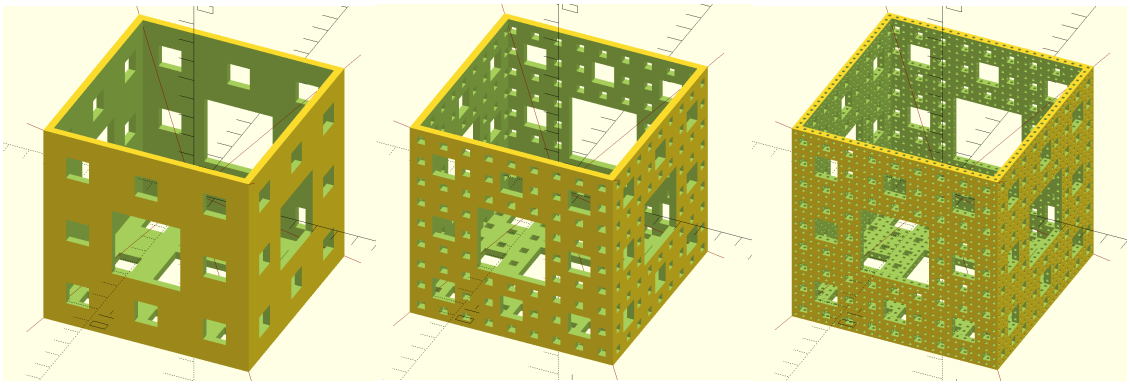


Figure 7. New design that generates hallow square hole by depth.

Description

i. Start from making one shape

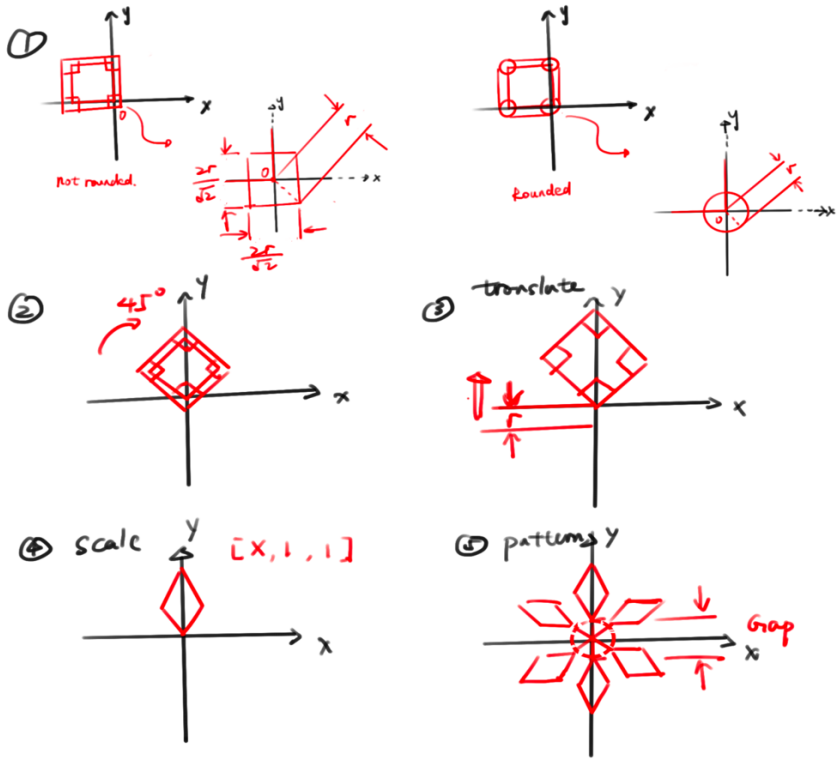


Figure 8. defining the shape.

Firstly, draw a cube and draw circle or square around its corners. If user want the shape is rounded, then the corner would be circle, otherwise, it would be square. If corner is rounded, the fillet radius would be the circle's radius, and if corner is not rounded, the side of the square would be $2 * radius/sqrt(2)$. Then the rotate and translate the shape to make into a flower pattern. The steps are shown above, and the code and its variable are shown below.

```
module draw_flower(gap,ratio,length, diamond_depth, rounded, corner_rad){
   for (i=[0:60:360]){
      rotate([0,0,i]){
         translate([gap/2,0,0]){
            draw_diamond(ratio,length, diamond_depth,rounded, corner_rad);
         }
      }
   }
}
```

Gap: the displacement between each diamond and the origin.
Ratio: the ratio between diamond's width and length.
Length: length of the diamond.
Diamond depth: extruded height of the diamond.
Rounded: a boolean that define weather the diamond is rounded.
Corner_rad: fillet radius.

ii. Map the flower

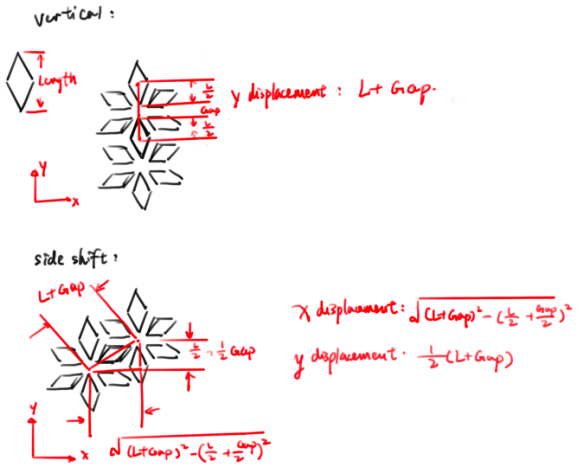


Figure 9. defining the shape.

After the shape of the flower is defined, we map the follower to make a pattern. And how to map the flower is defined in the picture above. Besides simply mapping the shape, I wanted to make it change with the iteration of the loop. The simplified logic is shown below, and the rest are listed in the Appendix.

```
if ( depth >= decay_depth ){
        depth -1
        Map the pattern without changing any parameters.
}
else if (( depth > 0 )&&( depth < decay_depth )){
        depth -1
        The length and the corner radius change with the depth of the loop
}
```

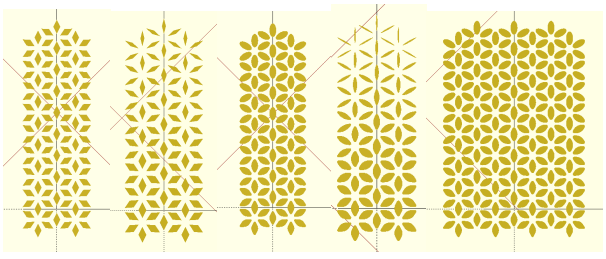


Figure 10. Pattern with different parameters.

iii. Extrude the pattern and cut it out from the lampshade

Extrude the pattern and use difference function to cut it out from the object.

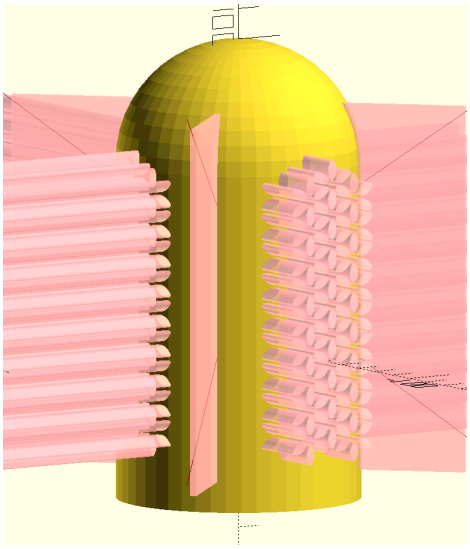


Figure 11. Extrusion of the patterns.

Finishing

Figure 12. clear surface finishing.

Presentation

Figure 13. Glamour photo printed lampshade with tealight, in dark

Figure 14. Glamour photo printed lampshade with tealight, in dark

Appendix I: Flower Lampshade code

```
// draw single diamond
module draw_diamond(ratio,length, diamond_depth,rounded, corner_rad){
   if (rounded == true){
      scale([1,ratio,1]){
         translate([corner_rad,0,0]){
            rotate([0,0,-45]){
               minkowski(){
                  {
                    cube([(length-2*corner_rad)/sqrt(2),(length-2*corner_rad)/sqrt(2),diamond_depth]);
                    cylinder(r=corner_rad,h=diamond_depth);
                  }
               }
            }
         }
      }
   }

   if (rounded == false) {
      scale([1,ratio,1]){
         translate([sqrt(2)*corner_rad/2,0,0]){
            rotate([0,0,-45]){
               minkowski(){
                  {
                    cube([(length-sqrt(2)*corner_rad)/sqrt(2),(length-sqrt(2)*corner_rad)/sqrt(2),diamond_depth]);
                    cube([corner_rad,corner_rad,diamond_depth],center=true);
                  }
               }
            }
         }
      }
   }
}
// rotate and translate the pattern
module draw_flower(gap,ratio,length, diamond_depth,rounded, corner_rad){
   for (i=[0:60:360]){
      rotate([0,0,i]){
         translate([gap/2,0,0]){
            draw_diamond(ratio,length, diamond_depth,rounded, corner_rad);
         }
      }
   }
}

// mapping the flowers vertically
module falling_flower(depth, initial_x,gap,ratio,length, diamond_depth,rounded, corner_rad){
   if (depth >= decay_depth){
      depth = depth-1;
      r = rounded;
      if (r == true){
         x = initial_x - (length+gap);
         translate([initial_x,0,0]){
            draw_flower(gap,ratio,length, diamond_depth,rounded, corner_rad);
         }
         falling_flower(depth, x ,gap,ratio,length, diamond_depth,rounded, corner_rad+0.5);
      }
      if (r == false){
         x = initial_x - (length+gap);
         translate([initial_x,0,0]){
            draw_flower(gap,ratio,length, diamond_depth,rounded, corner_rad);
```

```
            }
            falling_flower(depth, x, gap, ratio, length, diamond_depth, rounded, corner_rad);
        }
    }
    else if ((depth>0)&&(depth<decay_depth)){
        depth = depth-1;
        r = rounded;
        if (r == true){
            x = initial_x - (length+gap);
            translate([initial_x,0,0]){
                draw_flower(gap, ratio, length, diamond_depth, rounded, corner_rad);
            }
            falling_flower(depth, x, gap, ratio-width_decay_ratio, length, diamond_depth, rounded,
corner_rad+corner_rad_increase_num);
        }
        if (r == false){
            x = initial_x - (length+gap);
            translate([initial_x,0,0]){
                draw_flower(gap,ratio,length, diamond_depth,rounded, corner_rad);
            }
            falling_flower(depth, x ,gap,ratio-width_decay_ratio,length, diamond_depth,rounded, corner_rad +
corner_rad_increase_num);
        }
    }
}

// mapping the flowers pattern horizontally
module flower_pattern(num,depth, initial_x ,gap,ratio,length, diamond_depth,rounded, corner_rad){
    union(){
        for (i=[0:1:num]){
            if (i/2==round(i/2)){
                translate([-100,i*2*(length/2+gap/2)*cos(30),0]){
                    rotate([0,90,0]){
                        falling_flower(depth=depth, initial_x=initial_x,gap=gap,ratio=ratio,length=length,
diamond_depth=diamond_depth,rounded=rounded, corner_rad=corner_rad);
                    }
                }
                translate([-100,i*-2*(length/2+gap/2)*cos(30),0]){
                    rotate([0,90,0]){
                        falling_flower(depth=depth, initial_x=initial_x,gap=gap,ratio=ratio,length=length,
diamond_depth=diamond_depth,rounded=rounded, corner_rad=corner_rad);
                    }
                }
            }
            else {
                translate([-100,i*2*(length/2+gap/2)*cos(30),-(length+gap)/2]){
                    rotate([0,90,0]){
                        falling_flower(depth=depth, initial_x=initial_x,gap=gap,ratio=ratio,length=length,
diamond_depth=diamond_depth,rounded=rounded, corner_rad=corner_rad);
                    }
                }

                translate([-100,i*-2*(length/2+gap/2)*cos(30),-(length+gap)/2]){
                    rotate([0,90,0]){
                        falling_flower(depth=depth, initial_x=initial_x,gap=gap,ratio=ratio,length=length,
diamond_depth=diamond_depth,rounded=rounded, corner_rad=corner_rad);
                    }
                }
            }
        }
    }
}
```

```
// resolution
$fn = 50;
// depth of the pattern(total iteration of the pattern)
depth = 8;
// which step to shrink size of the pattern
decay_depth = 0;
// the ratio the width decay (width = (ratio - width_decay_ratio)*length)
width_decay_ratio = 0;
// increment of the corner radius
corner_rad_increase_num = 0;
// initial position to generate pattern
initial_x = 0;
// how much do those dimands spaced
gap = 10;
// width v.s length ratio
ratio = 0.5;
// length of the diamond
length = 18;
// how much to extrude the pattern
diamond_depth = 1000;
// fillet on or off
rounded = true;
// corner radius
corner_rad = 5;
// horizontal iteration of the pattern
num = 1;



// adjust the size of the pattern to make them fit the lampshade dimension
scaledown = 0.35;
// cut out the pattern
difference(){
    translate([0,0,-50]){
        union(){
            cylinder(h=110,r=40);
            translate([0,0,110]){
                sphere(r=40);
            }
        }
    }
    // unite the patterns
    #union(){
        rotate([0,0,90]){
            translate([-20,0,-20]){
                scale([scaledown,scaledown,scaledown]){
                    flower_pattern(num=num,depth=depth, initial_x=initial_x,gap=gap,ratio=ratio,length=length,
diamond_depth=diamond_depth,rounded=rounded, corner_rad=corner_rad);
                }
        }
        }
        translate([-50,0,-20]){
            scale([scaledown,scaledown,scaledown]){
                flower_pattern(num=num,depth=depth, initial_x=initial_x,gap=gap,ratio=ratio,length=length,
diamond_depth=diamond_depth,rounded=rounded, corner_rad=corner_rad);
            }
        }
        translate([0,0,20]){
            rotate([0,0,45]){
                cube([150,2,120],center=true);
            }
```

```
        }
        translate([0,0,20]){
            rotate([0,0,135]){
                cube([150,2,120],center=true);
            }
        }
// uncomment following to add a stamp on the front of the cylinder
//        translate([15,-30,-45]){
//            rotate([90,0,180]){
//                scale([0.1, 0.1, 0.2]){
//                    surface(file = "snow.png", center = true, invert = true);
//                }
//            }
//        }
    }
}
```

Appendix II: Cube Lampshade code

```
$fn = 10;
module draw_rectangule(depth,size,height,x,y){
   if (depth > 0){
      h = height;
      x1 = x;
      x2 = x1 + size/3;
      y1 = y;
      y2 = y1 + size/3;
      translate([x1,y1,0]){
         cube([size/3, size/3, h],center=true);
      }
      draw_rectangule(depth-1,size/3,h,x2,y2);
      draw_rectangule(depth-1,size/3,h,-x2,-y2);
      draw_rectangule(depth-1,size/3,h,x2,-y2);
      draw_rectangule(depth-1,size/3,h,-x2,y2);
      draw_rectangule(depth-1,size/3,h,x1,y2);
      draw_rectangule(depth-1,size/3,h,x1,-y2);
      draw_rectangule(depth-1,size/3,h,x2,y1);
      draw_rectangule(depth-1,size/3,h,-x2,y1);
   }
}

module recursive_cube(size,depth,x,y){
   difference(){
      cube(size,center=true);
      rotate([0,90,0]){
         draw_rectangule(depth,size,size*1.1,x,y);
      }
      rotate([90,0,0]){
         draw_rectangule(depth,size,size*1.1,x,y);
      }
      draw_rectangule(depth,size,size*1.1,x,y);
   }
}
module shell_cube(depth,size,side,extrusion_cut){
   difference(){
      union(){
         recursive_cube(size=size,depth=depth,x=0,y=0);
   // UNCOMMENT TO GENERATE ANOTHER CUBE ABOVE
   //       translate([0,0,10]){
   //          recursive_cube(size=10,depth=4,x=0,y=0);
   //       }
      }
      translate([0,0,size/2+0.07*size]){
         cube([side,side,extrusion_cut],center=true);
      }
   }
   // Show extrusion cut
   #translate([0,0,size/2+0.07*size]){
      cube([side,side,extrusion_cut],center=true);
   }
}
depth = 2;
size = 20;
side = size * 0.93;
extrusion_cut = size*2.1;

shell_cube(depth,size,side,extrusion_cut);
```