Procedural Chinese Lantern Using OpenSCAD

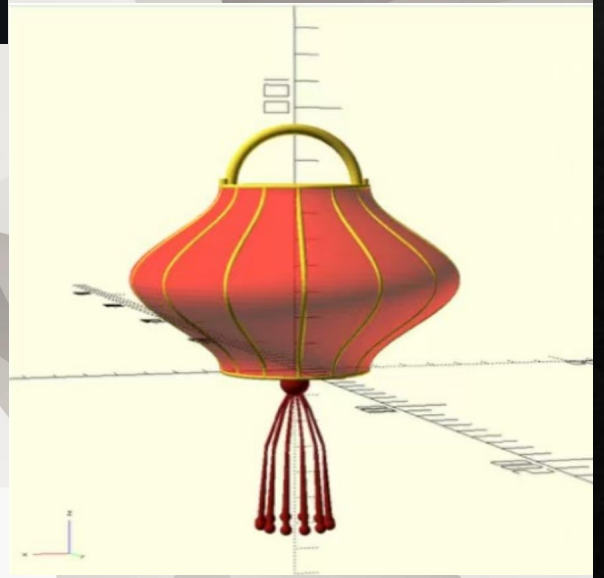
Group One

Hongcheng Pan

Sicheng xi

Yubo Wang

```
/* [Main Parameters] */
base_diameter = 50;       // Diameter at top and bottom (mm)
lantern_height = 70;      // Total height (mm)
paper_thickness = 0.8;    // Wall thickness (mm)

/* [Frame Settings] */
bulge_factor = 1.2;       // Middle bulge factor (1.0-1.5)
rib_count = 14;           // Number of ribs
rib_thickness = 2.0;      // Rib thickness (mm)

/* [Paper Rib Effect] */
rib_amplitude = 1.5;      // Paper bulge height between ribs (mm)
rib_width = 3.0;          // Width of each bulge (mm)

/* [Handle Settings] */
handle_height = 20;       // Handle arch height (mm)
handle_thickness = 4;     // Handle thickness (mm)

/* [Tassel Settings] */
tassel_length = 60;       // Tassel length (mm)
tassel_count = 12;        // Number of tassel strands

/* [Component Toggles] */
show_paper = true;        // Show paper shell
show_frame = true;        // Show internal frame
show_handle = true;       // Show top handle
show_tassel = true;       // Show bottom tassel

function radius_at_height(z) =
    let(
        t = z / lantern_height,
        bulge = bulge_factor * pow(cos(180 * (t - 0.5)), 2)
```



Project Overview Goals

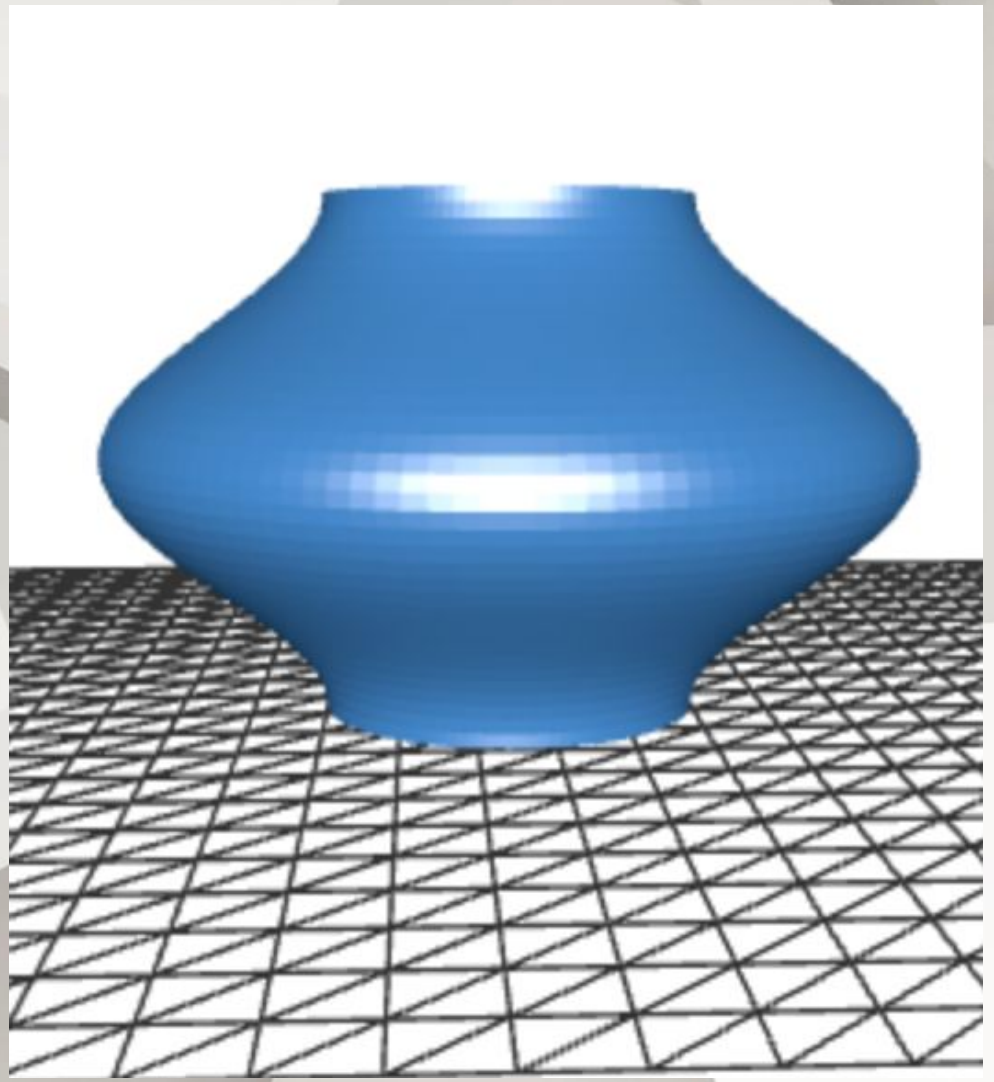
- This project demonstrates a fully procedural Chinese lantern model created using OpenSCAD. All geometry is generated through code, avoiding traditional 3D modeling software.
  The focus is on parametric control, modular structure, and script-driven rendering logic.
- **Key Objectives:**
- Translate mathematical logic into customizable geometry
- Achieve clean component separation (shell, frame, handle, tassel)
- Enable interactive parameter tuning (height, diameter, rib count, etc.)
- Provide a realistic visual with clear internal structure

Parametric Design & Radius Function

The lantern profile is defined using a custom function radius_at_height(z)
which controls curvature based on height and a configurable bulge factor.👇

```
function radius_at_height(z) =
    let(t = z / lantern_height,
        bulge = bulge_factor * pow(cos(180 * (t - 0.5)), 2))
    (base_diameter / 2) * (1 + bulge);
```

Key Parameters:
lantern_height
base_diameter
bulge_factorrib_count
rib_thickness
rib_amplitudepaper_thickness,
tassel_count
handle_height

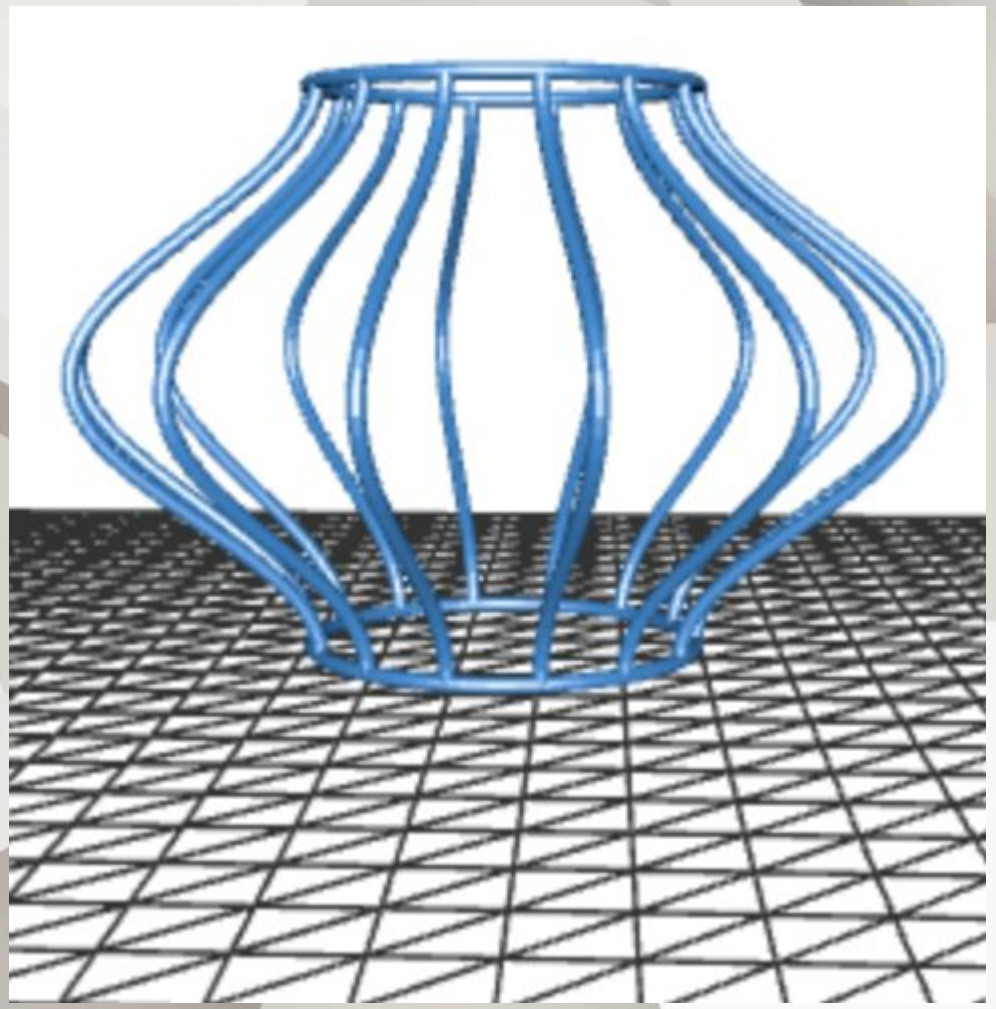


Shell & Frame Construction

The outer paper shell and internal frame are generated separately using rotate_extrude() and hull() operations.
Shell thickness is controlled via nested difference() of outer and inner profiles.
Ribs are constructed with connected spheres to maintain smooth curvature.

Techniques Used:

- 2D profile revolution (rotate_extrude)
- Boolean difference modeling (difference)
- Discrete rib lattice (hull + spheres)
- Paper rib texture simulated by extruded bulge shapes
- Circular decorations arranged via rotation loop

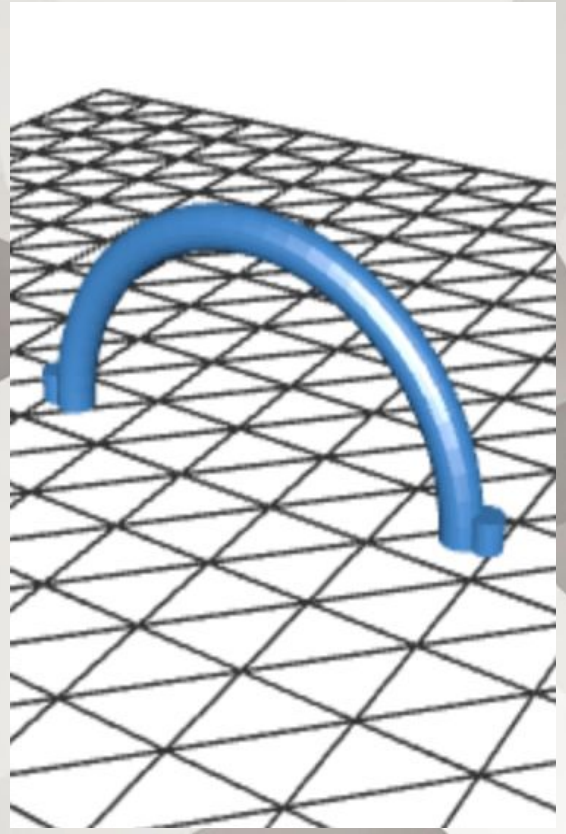


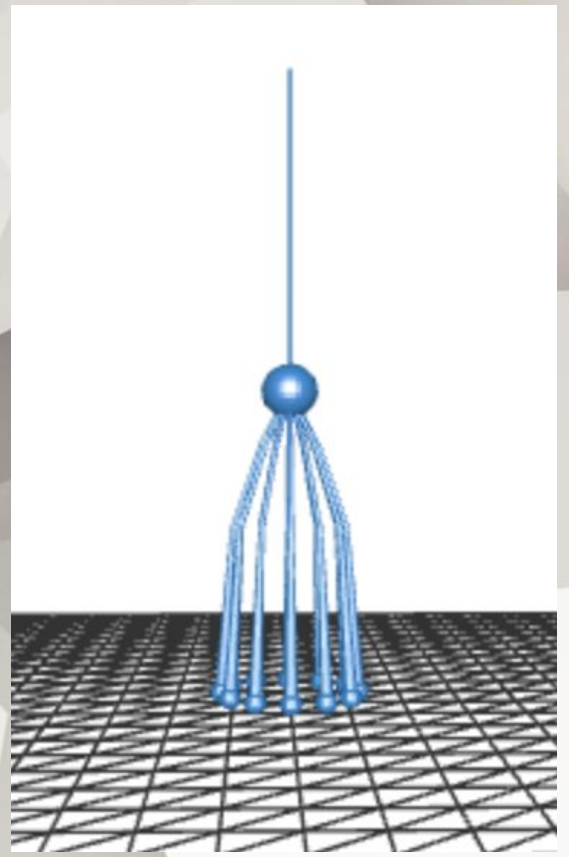
Handle & Tassel Modules

The top handle is modeled as a half torus using rotate_extrude(angle=180) with adjustable width and thickness.
The bottom tassel is constructed via a central sphere and multiple curved strands using chained spheres and hull().

Highlights:

- Parametric handle width from top radius
- Symmetrical tassel strands following Bezier-style paths
- Central gold thread using cylinder
- Individual strand ends are capped with decorative spheres





Rendering & Viewing

- For rendering and displaying the model with texture, I will be using HTML and three.js.
- For setting up the scene, the first tasks I did was setting up the perspective camera, the a directional and ambient lighting, and a basic WebGL renderer configured for full-screen canvas display. There was waso the default orbit controls for interactive rotation and zoom.
- The background color is set to a light cream tone, and the camera is positioned so that the whole lantern is in frame of the screen.
- The directional and ambient lights are set so that there are just enough shadows to show off the reflection, but no noticeable blind spots

```
//making the scene
const scene = new THREE.Scene();
scene.background = new THREE.Color(0xf7f6d5);

const camera = new THREE.PerspectiveCamera(70, window.innerWidth / window.innerHeight, 0.1, 1000);
camera.position.set(0, 0, 150);

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

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

//lights
scene.add(new THREE.AmbientLight(0xffffff, 0.75));
const dir = new THREE.DirectionalLight(0xffffff, 1);
dir.position.set(25, 30, 20);
scene.add(dir);
```

Texture & Shading

For shading, the paper and tassel will be getting its own texture map, while the frame and handle will be getting a standard gold color.

Each type of texture will have its own material:

- Paper: I want to create a rough, non-shiny surface that represents what paper looks like in real life, which a semi-transparent property to showcase its thinness.
- Tassel: The material will be similar to paper, but without the transparency and less rough.
- Gold: This texture will be very metallic, and less rough on the surface.

```
//lexture laoding and material creations
const texLoader = new THREE.TextureLoader();
const paperTex  = texLoader.load('RedPaper.jpg');
paperTex.wrapS = paperTex.wrapT = THREE.RepeatWrapping;  // tile if UVs > 1

const ropeTex   = texLoader.load('RopePaper.jpg');
ropeTex.wrapS = ropeTex.wrapT = THREE.RepeatWrapping;

const paperMat  = new THREE.MeshStandardMaterial({ map: paperTex,
  metalness: 0,
  roughness: 0.9,
  transparent: true,
  opacity: 0.65,
  side: THREE.DoubleSide
});
const goldMat   = new THREE.MeshStandardMaterial({ color: 0xffd062, metalness: 0.8, roughness: 0.3 });
const tasselMat = new THREE.MeshStandardMaterial({ map: ropeTex, metalness: 0, roughness: 0.6 });
```

Paper Texture

Tassel Texture

Texture Mapping

- With some help, I made a function that generates UV coordinates for STL models, so that the texture map can be correctly assigned onto the model.

```
// UV mapping helper
function addPlanarUVsXZ(geometry){
  if(geometry.attributes.uv) return; // already has UVs
  geometry.computeBoundingBox();
  const bb = geometry.boundingBox;
  const sizeX = bb.max.x - bb.min.x || 1;
  const sizeZ = bb.max.z - bb.min.z || 1;
  const pos = geometry.attributes.position;
  const uv = new Float32Array(pos.count * 2);
  for(let i=0;i<pos.count;i++){
    const x = pos.getX(i);
    const z = pos.getZ(i);
    uv[i*2]   = (x - bb.min.x) / sizeX;
    uv[i*2+1] = (z - bb.min.z) / sizeZ;
  }
  geometry.setAttribute('uv', new THREE.BufferAttribute(uv,2));
}
```

Model Loading & Animation

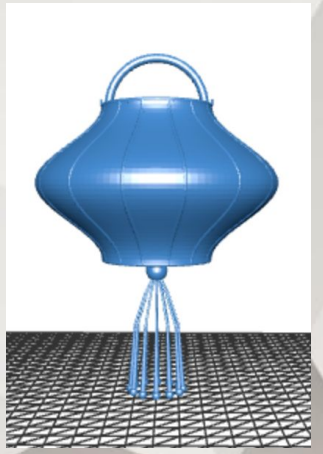
- The components of the lantern are then grouped together, and after applying its corresponding texture and material, it is rendered.
- For the animation, I did a simple rotation by the y axis.

```
//grouping comonents
const lanternGroup = new THREE.Group();
scene.add(lanternGroup);

const loader = new THREE.STLLoader();
function loadPart(path, material, addUV=false){
  loader.load(path, geo=>{
    geo.rotateX(-Math.PI/2);
    if(addUV) addPlanarUVsXZ(geo); // only needed for paper shell
    lanternGroup.add(new THREE.Mesh(geo, material));
  });
}

// load parts
loadPart('paper_shell.stl', paperMat, true);
loadPart('frame.stl', goldMat);
loadPart('handle.stl', goldMat);
loadPart('tassel.stl', tasselMat, true);
```

```
function animate(){
  requestAnimationFrame(animate);
  lanternGroup.rotation.y += 0.005;
  controls.update();
  renderer.render(scene,camera);
}
animate();
```

Conclusion & Reflection

This project successfully demonstrates a complete procedural modeling pipeline in OpenSCAD.
By translating mathematical functions into modular geometry, we explored the power of script-driven CAD thinking.



The following are the gains of each of us.

I think I learned the basic knowledge of computer graphics and the principle of generating images in the project. And some insights into game design and art. From the project, I have clarified my future bias direction and learned the basic application of openscad.

Through this project, I gained hands-on experience in procedural modeling using OpenSCAD.I learned how to translate mathematical logic into 3D geometry and how to construct modular structures using parameterized code.This process enhanced my understanding of CAD thinking and the power of script-based modeling.

I have learned how to apply texture and material onto a three dimentional model, and use three.js to render them in an HTML file. I also learned how to manipulate light so that the model displays its desired reflections.

Thanks For Your Listening!