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);
```

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.

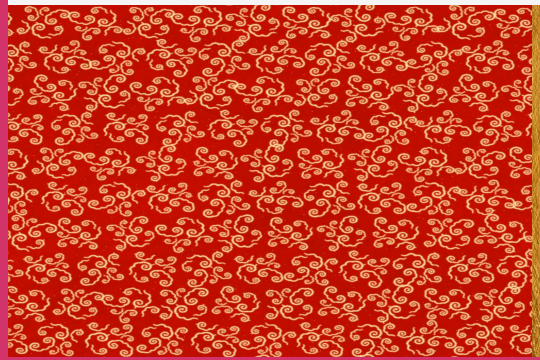
Each type of texture will have its own material:

- 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 });
```



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));
}
```

The components of the lantern are then grouped together, and after applying its corresponding texture and material, it is rendered.

```
//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);
```

For the animation, I did a simple rotation by the y axis.

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