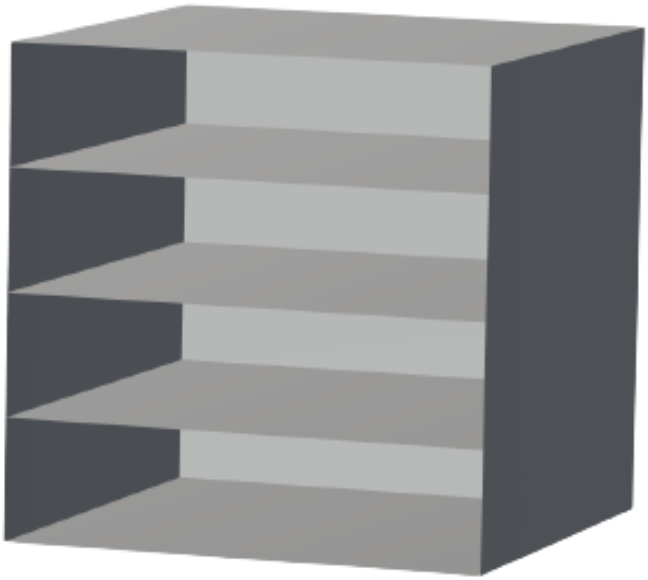
CABLE BOX



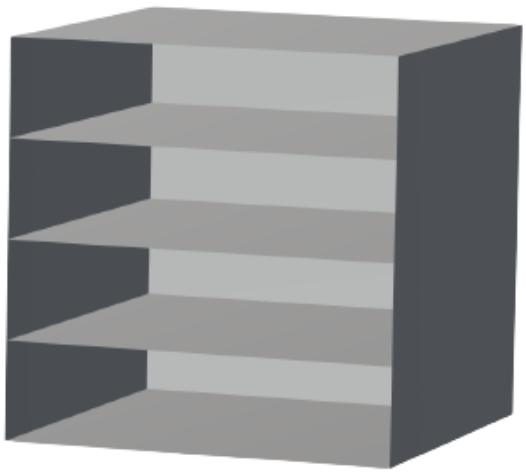
David Landete Expósito
Simón Brotons Cabrera
DM3D
11/04/2022

INDEX

References: Sources and references that you have consulted during the development of the project.

INTRODUCTION

We have developed a program to create a cable box. In the program you can define a few parameters to create a final model in STL format. The parameters you need to introduce are the width, height, depth and number of levels to manage the cables.
The basic model of a cable box with 4 levels, including floor. Using this box, you can pass the cable through the box so you can have an enter and exit hole.



The theorical concepts treated in this program is the generation of an STL file to create the model of the object. This STL file can be rendered and printed.

DEVELOPMENT

To develope the program we are using a basic HTML + JS + CSS combination.
The view is created with HTML and using js we can have a frontend function to send the parameters to the backend.
To create the backend, we have a basic server that receives a message with the parameters and generates the exit file in STL format.

The first thing to talk about is the creation of the **REST API** that we use to communicate the frontend and backend of our application.
Our main method, **generateSTL,** makes use of four parameters: width, height, depth and levels, to generate the final STL file.

```
server.get('/:param&:param2&:param3&:param4', (req, res) => {
  generateSTL(req.params.param, req.params.param2, req.params.param3, req.params.param4)
})
```

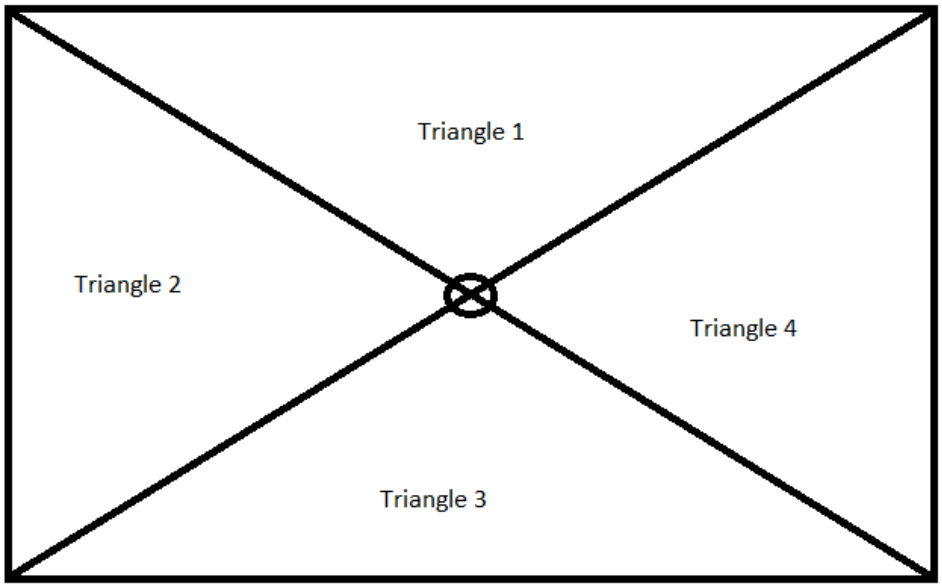
In the program, the first step is to create the outside box that will contain the levels to arrange the cables.

```
async function generateSTL(width, depth, height, nLayer) {
  stlString = " solid STL"

  generateVerticalFacet(0, height, depth);
  generateOppositeVerticalFacet(0, height, depth)

  generateVerticalFacet(width, height, depth);
  generateOppositeVerticalFacet(width, height, depth);

  generateHorizontalFacet(width, 0, depth);
  generateOppositeHorizontalFacet(width, 0, depth);

  generateHorizontalFacet(width, height, depth);
  generateOppositeHorizontalFacet(width, height, depth);
```

The next step is to calculate the distance between the levels of the box. We are using JavaScript so we need to use the function Number because if we don't use it, JavaScript will treat the addition as a String. If we don't use the function, the result of the addition would be '31' if we use '3' as the number of levels. In the other hand, when we use the function, the result is the required, '4'.

```
var layerHeight = (height) / (Number(nLayer) + 1)

for (let layer = 1; layer <= nLayer; layer++) {
  generateHorizontalFacet(width, (layer * layerHeight), depth);
  generateOppositeHorizontalFacet(width, (layer * layerHeight), depth);
}
```

In order to create a face, we need to calculate the triangles of the facet. We use the methods **generateVerticalFacet**, **generateOppositeVerticalFacet**, **generateHorizontalFacet** and **generateOppositeHorizontalFacet** to calculate the different facets. We need to create opposite facets because the object can be seen from different angles. If they're not calculated, we will see void where there should be a facet.
Each generate method, calls the function **facetTriangleString** four times, to generate the four triangles that represent the facet.




```
function generateVerticalFacet(width, height, depth) {
  let pointCenter = generatePoint(width, (height / 2), (depth) / 2);

  //First triangle
  let point1 = generatePoint(width, 0, 0);
  let point2 = generatePoint(width, 0, depth);
  facetTriangleString(point1, point2, pointCenter);

  //Second triangle
  let point3 = generatePoint(width, height, depth);
  facetTriangleString(point2, point3, pointCenter);

  //Third triangle
  let point4 = generatePoint(width, height, 0);
  facetTriangleString(point3, point4, pointCenter);

  //Fourth triangle
  facetTriangleString(point4, point1, pointCenter);
}
```

Each facet is composed by two points and a normal vector, all of them belonging to the facet.

```
function facetTriangleString(point1, point2, pointCenter) {
  let nVec = CalculateNVector(Points2Vector(point1, point2), Points2Vector(point1, pointCenter));
  stlString += `\n facet normal ${nVec.x} ${nVec.y} ${nVec.z}`
  stlString += `\n  outer loop`
  stlString += `\n   vertex ${point1.x} ${point1.y} ${point1.z}`
  stlString += `\n   vertex ${point2.x} ${point2.y} ${point2.z}`
  stlString += `\n   vertex ${pointCenter.x} ${pointCenter.y} ${pointCenter.z}`
  stlString += `\n  endloop`
  stlString += `\n endfacet`
}
```

During the creation of the facet, we call the method **calculateNVector**, where we multiply two vectors of the same facet to obtain the normal.

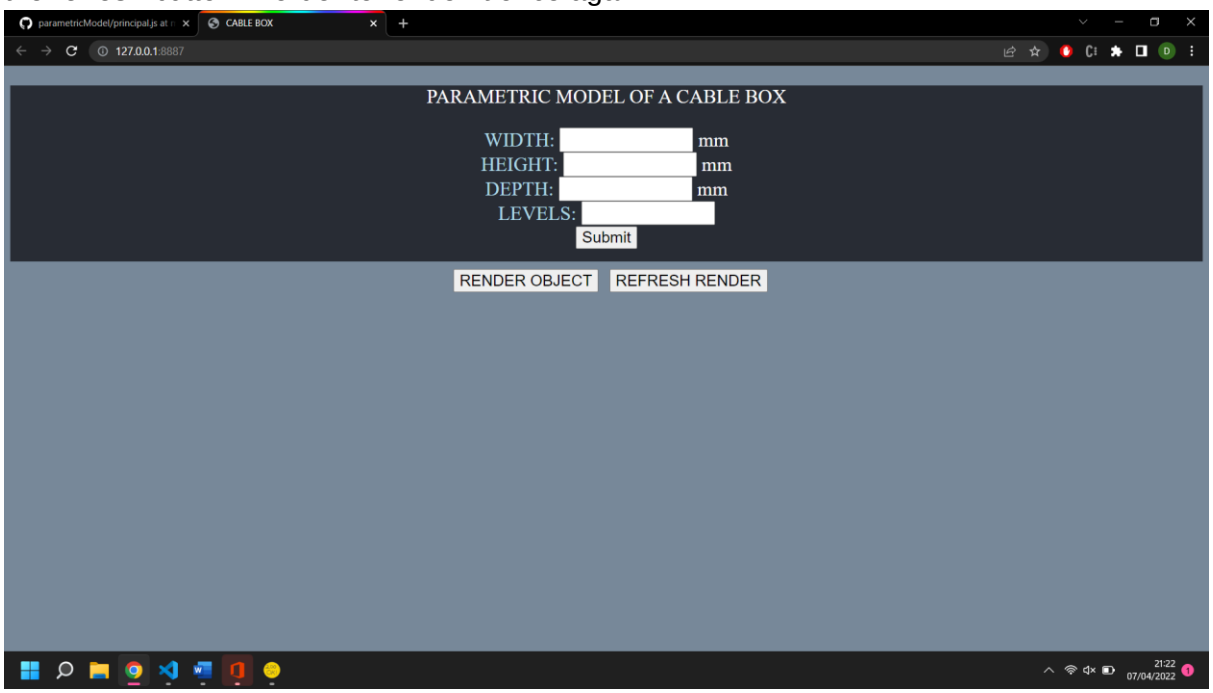
```
function calculateNVector(v1, v2) {
  let vec = new Object();
  vec.x = (v1.y * v2.z - v1.z * v2.y)
  vec.y = (v1.z * v2.x - v1.x * v2.z)
  vec.z = (v1.x * v2.y - v1.y * v2.x)
  return vec;
}
```

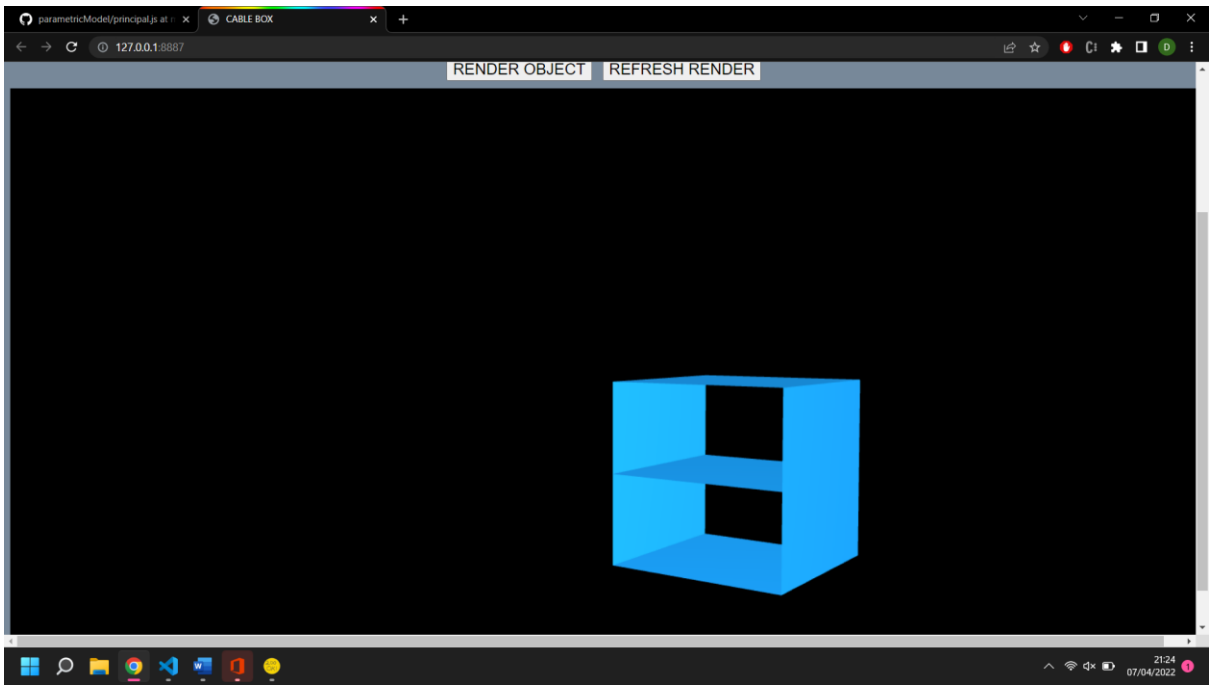
As you can see, the parameters of **facetTriangleString** do not include two vectors. In order to calculate them, we use the following function:

```
function points2Vector(p1, p2) {
  //p1 -> p2 (p2 - p1)
  let vec = new Object();
  vec.x = p2.x - p1.x;
  vec.y = p2.y - p1.y;
  vec.z = p2.z - p1.z;
  return vec;
}
```

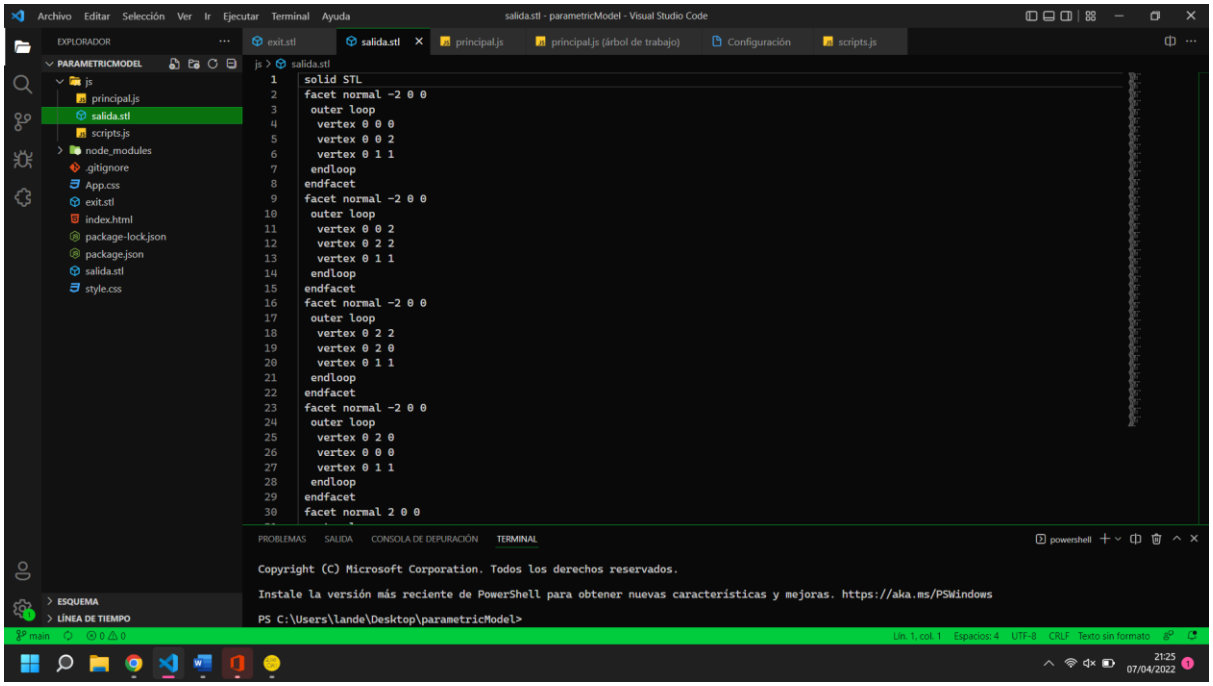
RESULTS

As a result of the development, we have a web divided in two zones. One that asks for input parameters (width, height, depth and levels) and a second one where a model is shown depending on the parameters that the user submitted.
Given the scenario where the user wants to modify the object, the user would have to press the refresh button in order to render it once again.





This is an example of a cable box with one level.



In this picture, we can see the generated STL file.

PERFORMANCE

We divided the tasks of the project so that we would end up doing 50% of the work.

REFERENCES

Fetch method front to back communication.
https://www.youtube.com/watch?v=2Xm9P_tXtK8&ab_channel=CarlosAzaustre-AprendeJavaScript

STL format and normal vector calculation.
Polygon Mesh Modelling pdf
https://poliformat.upv.es/access/content/group/GRA_11640_2021/Modules/2.%20Geometric%20Modelling/Polygon%20Mesh%20Modeling.pdf

Render a stl object from an html file.
https://stackoverflow.com/questions/12880980/need-js-and-html-example-for-displaying-stl-3d-objects-in-a-web-page