A Space-Themed Platformer

"Houston, We Had Problems"

-Logan Margo and Kieran Kim-Murphy

Table of contents

01 Project Overview

The concept and how it evolved

02 Technologies

New tools that we learned

03 Objectives

Our goals for this project

04 Implementation

Techniques and algorithms used

05 Demo

Our work so far

06 Recap

What we learned

Project Overview

01

Fall-Guys-esque platformer

02

Solar-System Hopper?

03

Procedural Planets!?!

(uh oh)

Technologies

Babylon.js

It’s like Three JS but the name is different

Vue.js

Handles pages and components

Vite.js

Build tool

Objectives

Learn new tech

- BabylonJS
- Vue and Vite

Scale

Assemble a larger-scale graphics project from the ground up.

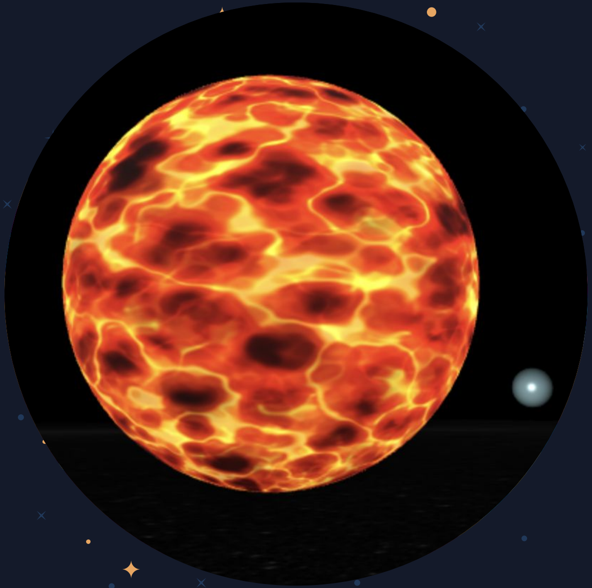
Space Themed

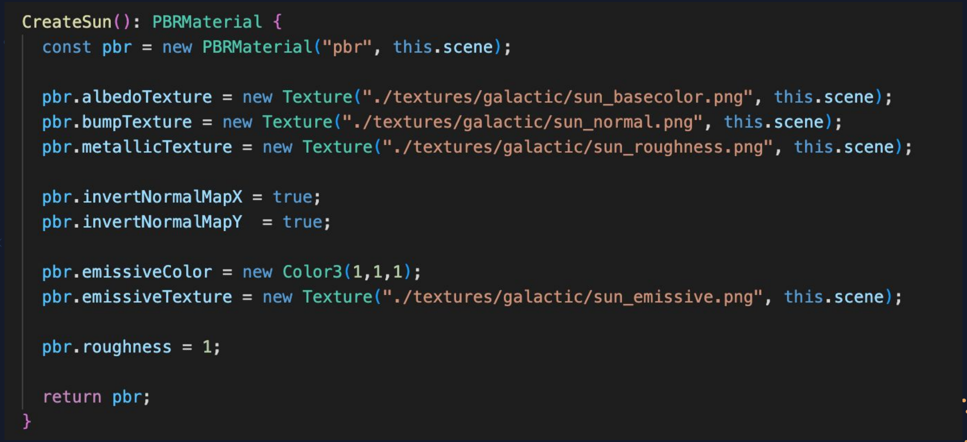
- Space is cool
- Lots of graphics opportunities

Concepts

Explore new concepts and algorithms

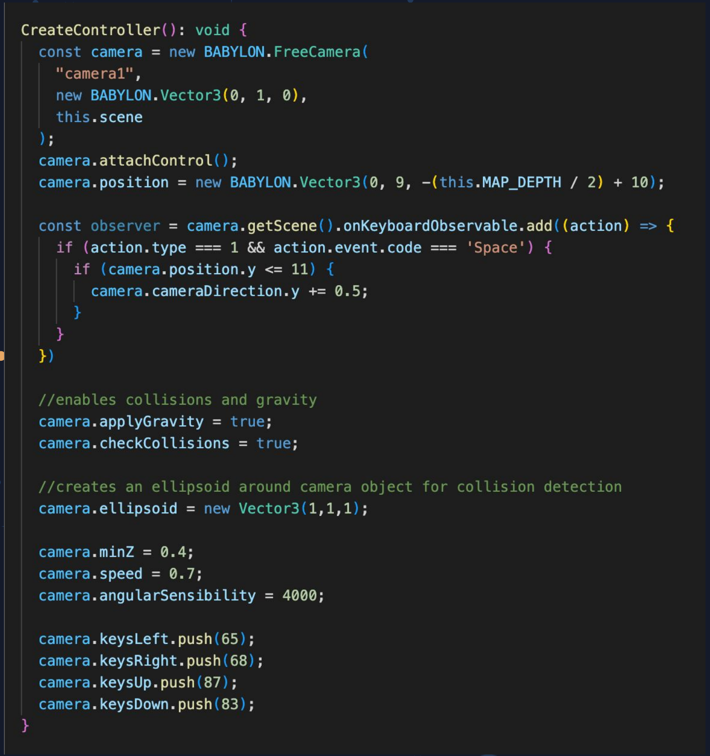
Emissive Textures





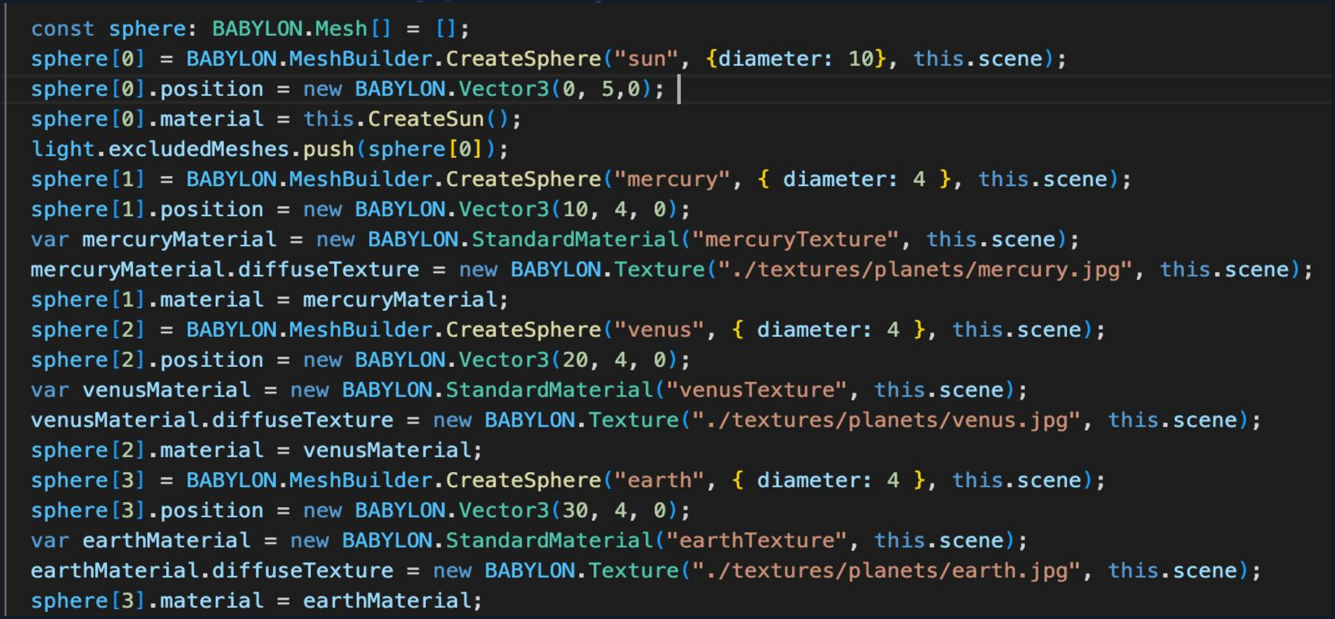
```
CreateSun(): PBRMaterial {
  const pbr = new PBRMaterial("pbr", this.scene);

  pbr.albedoTexture = new Texture("./textures/galactic/sun_basecolor.png", this.scene);
  pbr.bumpTexture = new Texture("./textures/galactic/sun_normal.png", this.scene);
  pbr.metallicTexture = new Texture("./textures/galactic/sun_roughness.png", this.scene);

  pbr.invertNormalMapX = true;
  pbr.invertNormalMapY  = true;

  pbr.emissiveColor = new Color3(1,1,1);
  pbr.emissiveTexture = new Texture("./textures/galactic/sun_emissive.png", this.scene);

  pbr.roughness = 1;

  return pbr;
}
```

First Person Camera Controls



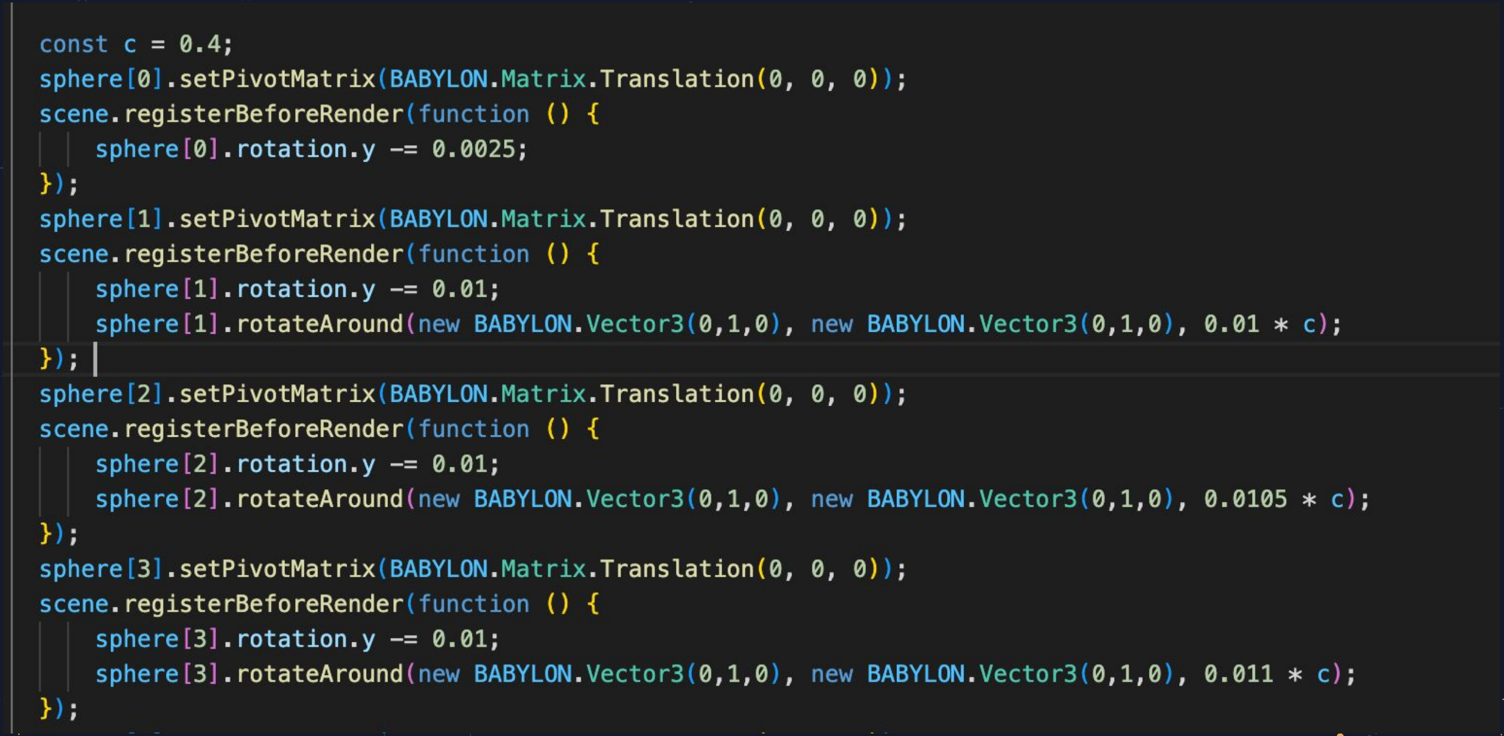
```
CreateController(): void {
  const camera = new BABYLON.FreeCamera(
    "camera1",
    new BABYLON.Vector3(0, 1, 0),
    this.scene
  );
  camera.attachControl();
  camera.position = new BABYLON.Vector3(0, 9, -(this.MAP_DEPTH / 2) + 10);

  const observer = camera.getScene().onKeyboardObservable.add((action) => {
    if (action.type === 1 && action.event.code === 'Space') {
      if (camera.position.y <= 11) {
        camera.cameraDirection.y += 0.5;
      }
    }
  })

  //enables collisions and gravity
  camera.applyGravity = true;
  camera.checkCollisions = true;

  //creates an ellipsoid around camera object for collision detection
  camera.ellipsoid = new Vector3(1,1,1);

  camera.minZ = 0.4;
  camera.speed = 0.7;
  camera.angularSensibility = 4000;

  camera.keysLeft.push(65);
  camera.keysRight.push(68);
  camera.keysUp.push(87);
  camera.keysDown.push(83);
}
```

Solar Scene Implementation



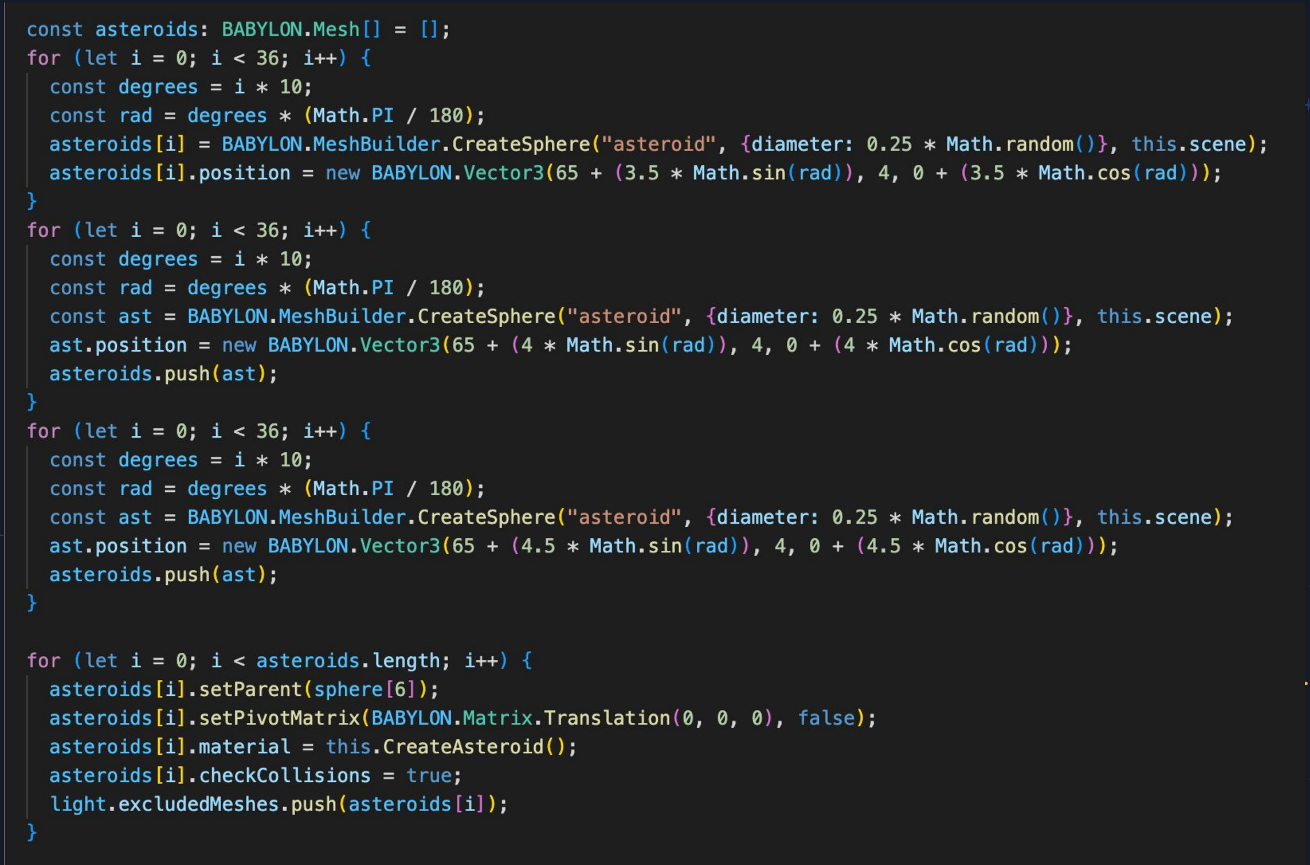
```
const sphere: BABYLON.Mesh[] = [];
sphere[0] = BABYLON.MeshBuilder.CreateSphere("sun", {diameter: 10}, this.scene);
sphere[0].position = new BABYLON.Vector3(0, 5,0);
sphere[0].material = this.CreateSun();
light.excludedMeshes.push(sphere[0]);
sphere[1] = BABYLON.MeshBuilder.CreateSphere("mercury", { diameter: 4 }, this.scene);
sphere[1].position = new BABYLON.Vector3(10, 4, 0);
var mercuryMaterial = new BABYLON.StandardMaterial("mercuryTexture", this.scene);
mercuryMaterial.diffuseTexture = new BABYLON.Texture("./textures/planets/mercury.jpg", this.scene);
sphere[1].material = mercuryMaterial;
sphere[2] = BABYLON.MeshBuilder.CreateSphere("venus", { diameter: 4 }, this.scene);
sphere[2].position = new BABYLON.Vector3(20, 4, 0);
var venusMaterial = new BABYLON.StandardMaterial("venusTexture", this.scene);
venusMaterial.diffuseTexture = new BABYLON.Texture("./textures/planets/venus.jpg", this.scene);
sphere[2].material = venusMaterial;
sphere[3] = BABYLON.MeshBuilder.CreateSphere("earth", { diameter: 4 }, this.scene);
sphere[3].position = new BABYLON.Vector3(30, 4, 0);
var earthMaterial = new BABYLON.StandardMaterial("earthTexture", this.scene);
earthMaterial.diffuseTexture = new BABYLON.Texture("./textures/planets/earth.jpg", this.scene);
sphere[3].material = earthMaterial;
```

Solar Scene Implementation



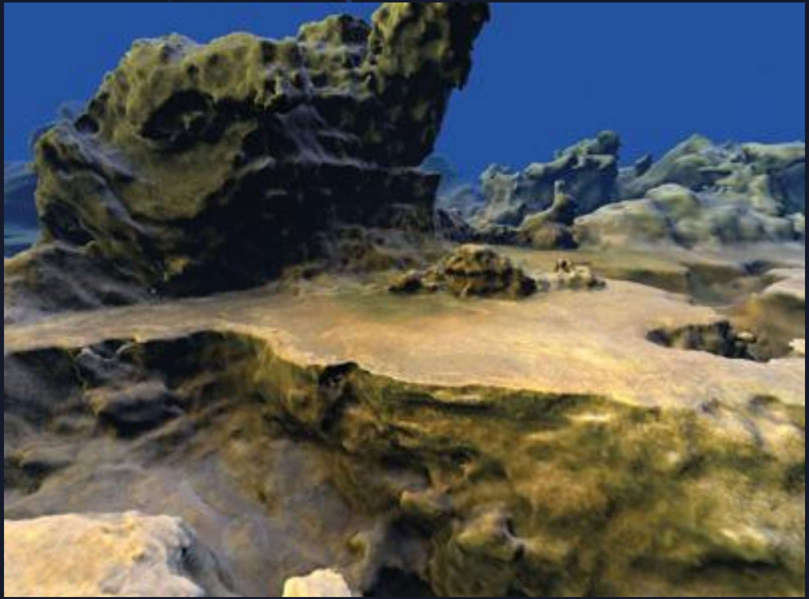
```
const c = 0.4;
sphere[0].setPivotMatrix(BABYLON.Matrix.Translation(0, 0, 0));
scene.registerBeforeRender(function () {
    sphere[0].rotation.y -= 0.0025;
});
sphere[1].setPivotMatrix(BABYLON.Matrix.Translation(0, 0, 0));
scene.registerBeforeRender(function () {
    sphere[1].rotation.y -= 0.01;
    sphere[1].rotateAround(new BABYLON.Vector3(0,1,0), new BABYLON.Vector3(0,1,0), 0.01 * c);
});
sphere[2].setPivotMatrix(BABYLON.Matrix.Translation(0, 0, 0));
scene.registerBeforeRender(function () {
    sphere[2].rotation.y -= 0.01;
    sphere[2].rotateAround(new BABYLON.Vector3(0,1,0), new BABYLON.Vector3(0,1,0), 0.0105 * c);
});
sphere[3].setPivotMatrix(BABYLON.Matrix.Translation(0, 0, 0));
scene.registerBeforeRender(function () {
    sphere[3].rotation.y -= 0.01;
    sphere[3].rotateAround(new BABYLON.Vector3(0,1,0), new BABYLON.Vector3(0,1,0), 0.011 * c);
});
```

Solar Scene Implementation



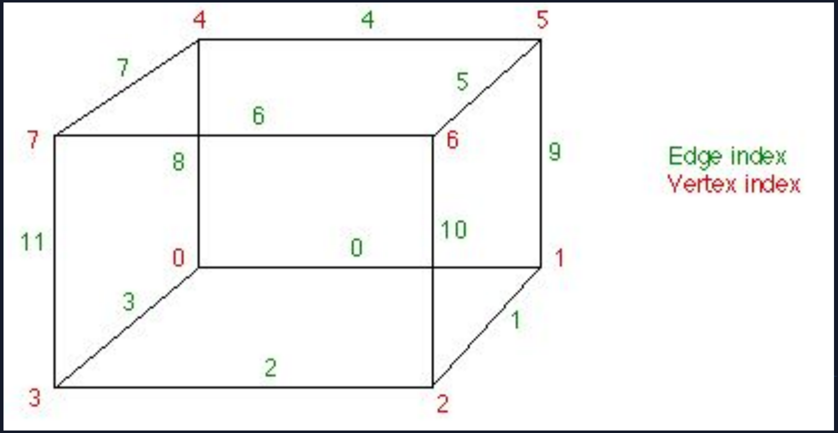
```
const asteroids: BABYLON.Mesh[] = [];
for (let i = 0; i < 36; i++) {
  const degrees = i * 10;
  const rad = degrees * (Math.PI / 180);
  asteroids[i] = BABYLON.MeshBuilder.CreateSphere("asteroid", {diameter: 0.25 * Math.random()}, this.scene);
  asteroids[i].position = new BABYLON.Vector3(65 + (3.5 * Math.sin(rad)), 4, 0 + (3.5 * Math.cos(rad)));
}
for (let i = 0; i < 36; i++) {
  const degrees = i * 10;
  const rad = degrees * (Math.PI / 180);
  const ast = BABYLON.MeshBuilder.CreateSphere("asteroid", {diameter: 0.25 * Math.random()}, this.scene);
  ast.position = new BABYLON.Vector3(65 + (4 * Math.sin(rad)), 4, 0 + (4 * Math.cos(rad)));
  asteroids.push(ast);
}
for (let i = 0; i < 36; i++) {
  const degrees = i * 10;
  const rad = degrees * (Math.PI / 180);
  const ast = BABYLON.MeshBuilder.CreateSphere("asteroid", {diameter: 0.25 * Math.random()}, this.scene);
  ast.position = new BABYLON.Vector3(65 + (4.5 * Math.sin(rad)), 4, 0 + (4.5 * Math.cos(rad)));
  asteroids.push(ast);
}

for (let i = 0; i < asteroids.length; i++) {
  asteroids[i].setParent(sphere[6]);
  asteroids[i].setPivotMatrix(BABYLON.Matrix.Translation(0, 0, 0), false);
  asteroids[i].material = this.CreateAsteroid();
  asteroids[i].checkCollisions = true;
  light.excludedMeshes.push(asteroids[i]);
}
```

Terrain Generation



Source: Nvidia [2]

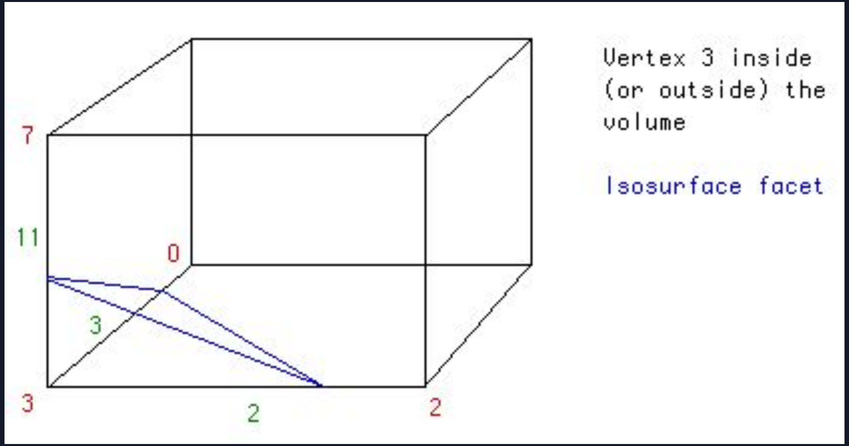
Marching Cubes



Source: Bourke [1]

$$f(x, y, z) \rightarrow value$$

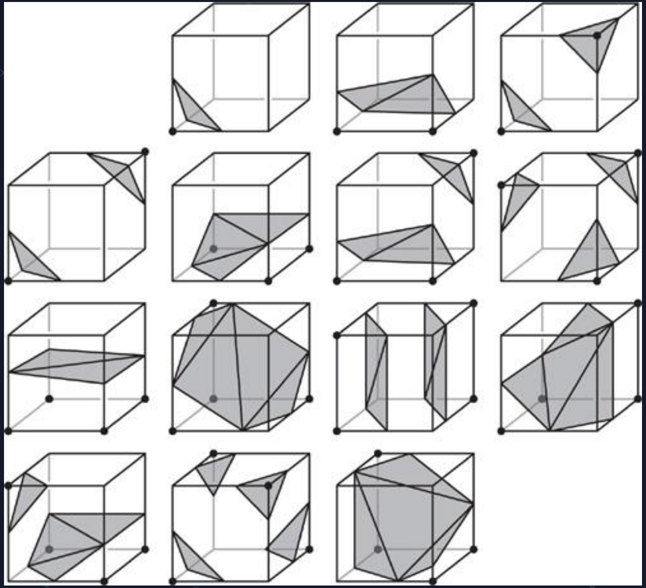
Marching Cubes



Source: Bourke [1]

$$f(x, y, z) \rightarrow value$$

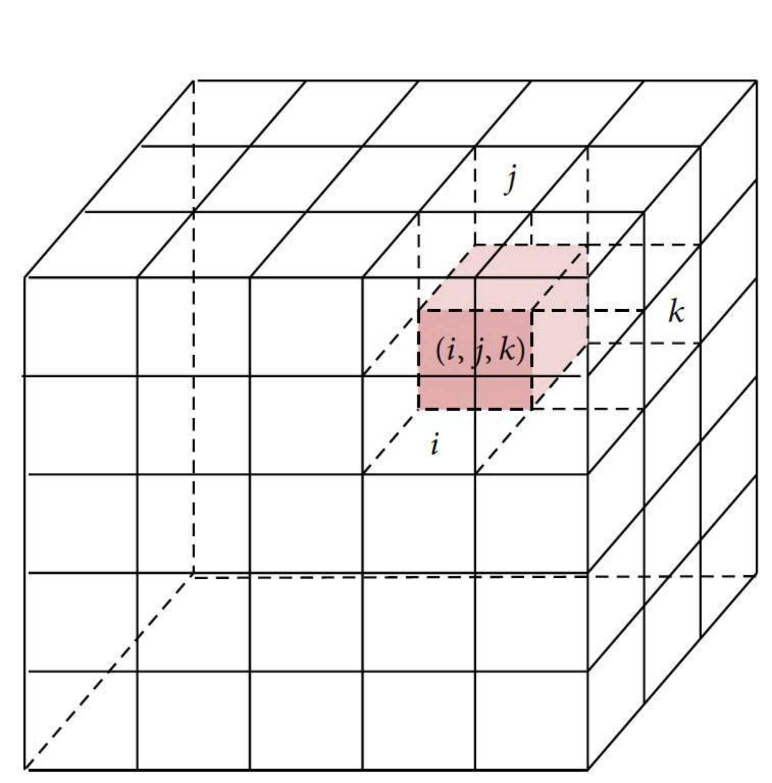
Marching Cubes



Source: Nvidia [2]

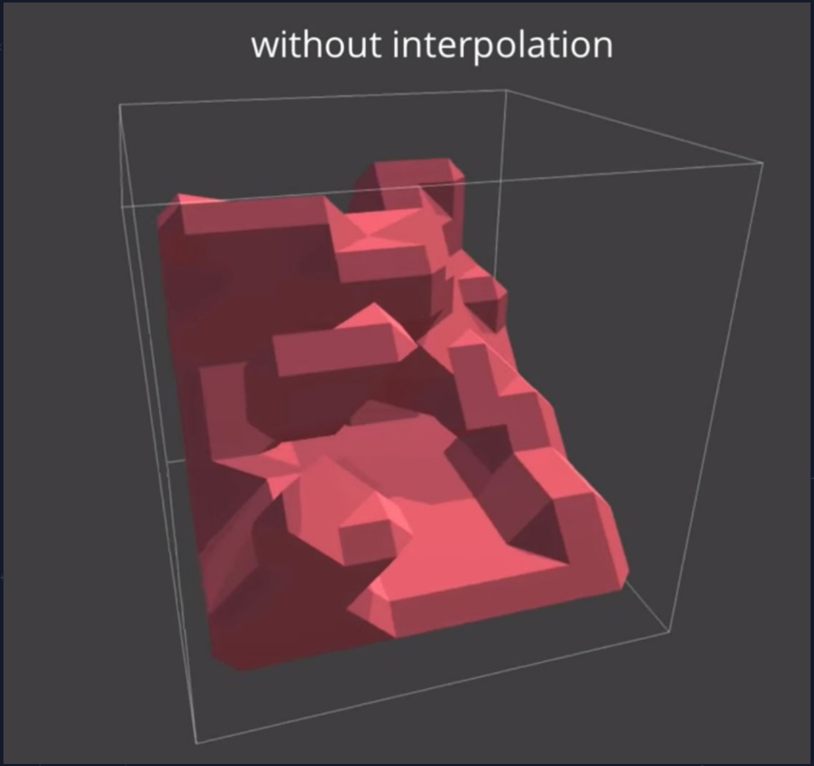
$$f(x, y, z) \rightarrow value$$

Marching Cubes



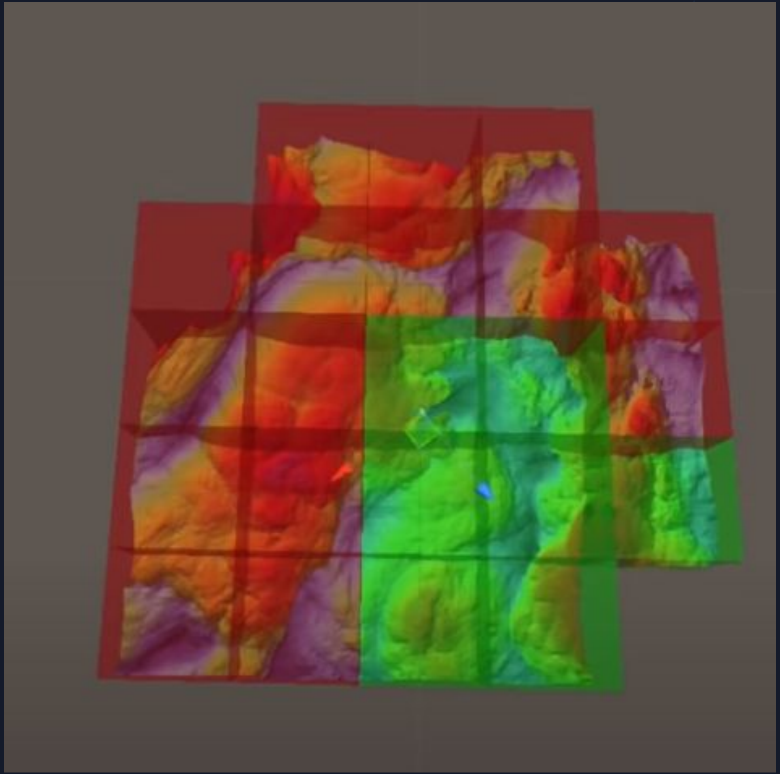
Source: Kruger [6]

Marching Cubes



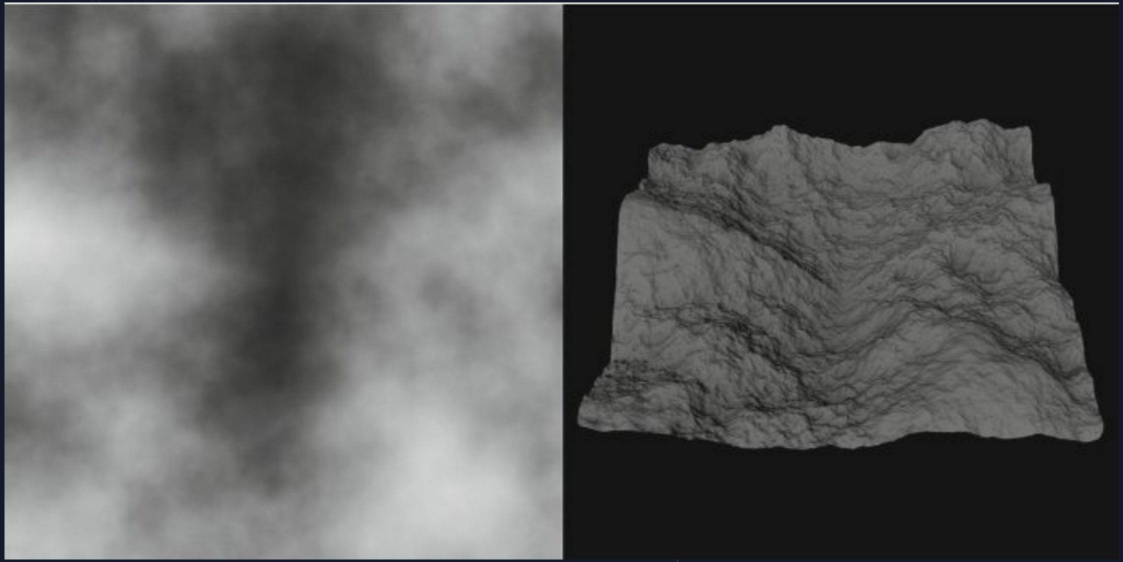
Source: Lague [4]

Marching Cubes



Source: Lague [4]

Marching Cubes



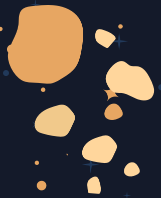
Source: Eck and Lamers [7]

$$f(x, y, z) \to value$$

Demo

Recap!

What we learned



Thank you!

Questions?

Project Sources

[1] Paul Bourke. 1995. Polygonising a scalar field. Blog. “http://paulbourke.net/geometry/polygonise/” Accessed May, 2022.

[2] Ryan Geiss. Generating Complex Procedural Terrains Using the GPU. Blog. “https://developer.nvidia.com/gpugems/gpugems3/part-i-geometry/chapter-1-generating-complex-procedural-terrains-using-gpu” Accessed May, 2022.

[3] Matthew Fisher. 2014. Marching Cubes. Blog. “https://graphics.stanford.edu/~mdfisher/MarchingCubes.html” Accessed May, 2022.

[4] Sebastian Lague. 2019. Coding Adventure: Marching Cubes. Video. “https://www.youtube.com/watch?v=M3iI2l0ltbE” Accessed May, 2022.

[5] Babylon.js Documentation. “https://doc.babylonjs.com” Accessed April 2022.

[6] Berti Kruger. 2020. Forum. “https://discourse.vtk.org/t/vtkmarchingcubes-vtkdiscretemarchingcubes-does-not-produce-a-closed-mesh-surface/4312” Accessed May, 2022.

[7] Wim Eck and Maarten Lamers. Biological Content Generation: Evolving Game Terrains Through Living Organisms. 2015. Book.