



A Gentle Introduction Babylon.js Overview and Worked Example

Carl Bateman
WebGL Workshop

Table of Contents

Part 1. Agenda	2
Part 2. Babylonjs.com	3
Part 3. Preparation	4
Glitch.com.....	5
index.html	5
style.css	6
script.js	6
Shapes.....	6
Mesh vs MeshBuilder.....	7
Part 4. Resources	8
Examples.....	8
Playground.....	8
Create Your Own Shader	8
Documentation.....	8
Forum	8
Source	8
Sandbox	8
Part 5. It's Editing Time!.....	9
Moar Preparation (Sorry)	9
Remix!.....	9
Take (Interactive) Control!	9
Add a Pile of Boxes	9
Functions	10
Part 6. Animation.....	11
Position	11
Scale.....	11
Rotation	11
Color	11
Play time	11
Part 7. Vertices.....	13
Remix Reminder!.....	13
Fun Stuff or Something A Bit Different, SDF.....	13
Part 8. More advanced stuff.....	17

Part 1. Agenda

Well, take a look at the table of contents.

- Intro to Babylon.js:
 - look at the Babylon.js website
 - look at some of resources available
- Review the basic sample app from the Babylon.js web site
- Using Glitch.com for editing
- Make a few minor changes:
 - material/colour
 - control
- Properties to change:
 - position
 - rotation
 - scale
 - colour
- Factors/functions to derive values to use to control the above
 - step
 - cos, sin
 - sqrt
 - abs
 - PI

Part 2. Babylonjs.com



WebGL. Simple. Powerful.
A complete JavaScript framework for building 3D games and experiences with HTML5, WebGL, WebVR and Web Audio

[TRY](#) [DOWNLOAD](#) [GITHUB](#)

[Examples](#) [Specifications](#) [Documentation](#) [Tutorials](#) [Forum](#) [Support](#) [CDDG](#) [Sponsors](#)

Demos

Filtered: All



Apartment Configurator



Bronchoscopy Simulation



Brilliant man



DefVizion Motion Ring Maps



LIGHT SPEED READY



JUMPY



INFINITE TERRAIN



Web Shelf

Specifications

Main features

- Transparent WebGL 1.0 / WebGL 2.0 support
- Complete scene graph with lights, cameras, materials, meshes, animations, audio & videos
- Native collision engine
- Easy to use full featured viewer
- Physics engine (thanks to three.js & cannon.js)
- Scene picking
- Supports left and right handed systems
- Animating
- Animation engine
- Particles (both CPU and GPU) & Solid Particles Systems
- Spines and 2D layers
- Complete audio engine based on Web Audio
- Hardware accelerated GUI
- Behaviors

Works on all WebGL platforms via a specific modern shader architecture and native touch support: E1 (V8) Edge, Chrome, Firefox, Opera, Safari, iOS (iPad/iPhone), Android, Windows Phone 8.1/Mobile 10, Firefox OS, Xbox One

Optimizations

- Frustum culling
- Sub meshes clipping
- Hardware scaling
- Occlusion culling
- Selection culling
- Offline mode (Assets saved in IndexedDB)
- Instanced loading
- Binary compressed format
- Hardware accelerated statistics
- Automatic scene statistics
- LOD (Level Of Detail)
- Collisions on Web Workers
- Meshes merging

Shaders / Rendering

- **Physically Based Rendering (PBR)**
- **Standard material** is a per pixel material that supports:
 - Diffuse lighting and texture
 - Ambient lighting and texture
 - Specular lighting

- Opacity texture
- Refraction texture (Sphere, plane, cube, projection and rectangular)
- Mirror texture
- Emissive texture
- Specular texture
- Bump texture
- Lightmap texture
- Unlimited lights (spots, directionals, spots, hemispheres)
 - Light texture projection
- Custom materials
- Custom shaders
- Skelton
- Vertex color
- Up to 8 bones per vertex
- Morphing
- Fresnel term for diffuse, opacity, emissive and refraction

Procedural features library

- Materials library

Special FX

- Fog
- Alpha blending
- Alpha testing
- SSR (refracting)
- Fullscreen mode
- Shadow Maps and Exponential Shadow Maps (including soft shadows with PCF and PCSS)
- Rendering layers
- Post processing (Blur, reflection, black/white, color correction, Vign, SSAO (including WebGL version), God Rays (V.S.), HDR, DOF (Depth Of Field), tonemapping, sharpen, gain (image processing), custom...)
- Lens flares
- Reflection Probe
- Multi-cameras
- Edges renderer
- Instanlights (object)
- Glow lines

Textures

- Render target textures
- Dynamic textures (2D canvas)
- Video textures (including from webcams)
- Compressed (DDS, KTX) textures & TGA

Cameras

- Universal camera (keyboard/touch/gamepad)
- Arc rotate camera
- Free camera
- Touch camera
- Virtual/Artychica camera
- Gamepad camera
- VR Device (Orientation camera for Cardboard)
- WebGL Camera VII (Emergency linker WebGL video done)
- Anaglyph camera
- Follow camera

Extremely simple to use!

Meshes

- Mesh cloning
- Dynamic meshes
- Instanced meshes
- Bones
- Constructive Solid Geometries
- Parametric shapes (Sphere, tubes, etc.)
- World targets

Exporters & Tooling

-  blender
-  unity
-  3DS MAX
-  MAYA
-  Clara.io

- Exporters for OBJ & glTF
- Importers for glTF, STL & OBJ
- Support for drag/drop
- Debug layer
- Assets Manager
- Assets Builder

Lead developers: David Catuhe - David Foussard - Sebastien Vandercheruix

Core contributors: Cédric Beaumont - Adam Roemer - Jérôme Boussouff - Julien Chenuard - Erwan Marzloff - Bastien Weber

3D Artist: Michel Rousseau

3rd Party assets: Carille, Joly and Enoplos - Train scene: Remald Roussel and Progris

Developed by:  netlify

[Like](#) [Share](#) It's possible this site. Be the first to give feedback

Part 3. Preparation

<https://doc.babylonjs.com/>

The Babylon.js site has the following sample HTML/JavaScript to create a simple scene, a sphere above a plane.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html" charset="utf-8"/>
  <title>Babylon - Getting Started</title>
  <!-- Link to the Last version of BabylonJS -->
  <script src="https://cdn.babylonjs.com/babylon.js"></script>
  <style>
    html, body {
      overflow: hidden;
      width : 100%;
      height : 100%;
      margin : 0;
      padding : 0;
    }

    #renderCanvas {
      width : 100%;
      height : 100%;
      touch-action: none;
    }
  </style>
</head>
<body>
  <canvas id="renderCanvas"></canvas>
  <script>
    window.addEventListener('DOMContentLoaded', function(){
      // get the canvas DOM element
      var canvas = document.getElementById('renderCanvas');

      // Load the 3D engine
      var engine = new BABYLON.Engine(canvas, true);

      // createScene function that creates and return the scene
      var createScene = function(){
        // create a basic BJS Scene object
        var scene = new BABYLON.Scene(engine);

        // create a FreeCamera, and set its position to (x:0, y:5, z:-10)
        var camera = new BABYLON.FreeCamera('camera1', new BABYLON.Vector3(0, 5,-10), scene);

        // target the camera to scene origin
        camera.setTarget(BABYLON.Vector3.Zero());

        // attach the camera to the canvas
        camera.attachControl(canvas, false);

        // create a basic light, aiming 0,1,0 - meaning, to the sky
        var light = new BABYLON.HemisphericLight('light1', new BABYLON.Vector3(0,1,0), scene);

        // create a built-in "sphere" shape; its constructor takes 6 params: name, segment, diameter, scene,
        // updatable, sideOrientation
        var sphere = BABYLON.Mesh.CreateSphere('sphere1', 16, 2, scene);

        // move the sphere upward 1/2 of its height
        sphere.position.y = 1;

        // create a built-in "ground" shape;
        var ground = BABYLON.Mesh.CreateGround('ground1', 6, 6, 2, scene);

        // return the created scene
      };
    });
  </script>
</body>
</html>
```

```

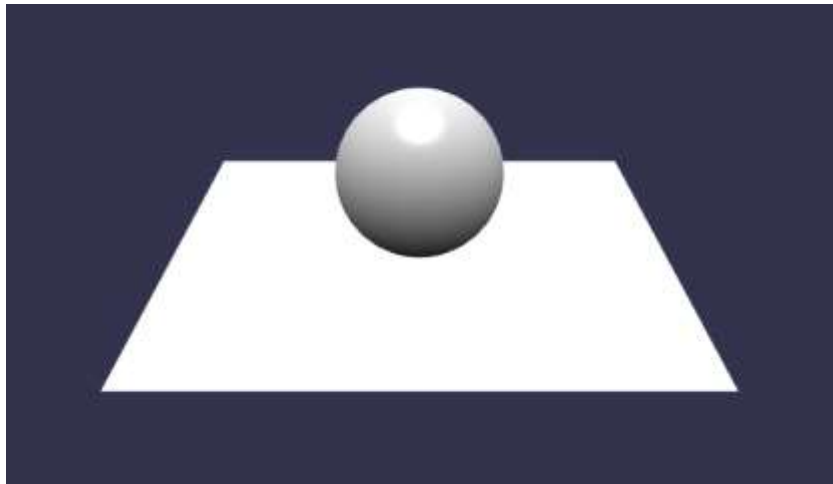
    return scene;
  }

  // call the createScene function
  var scene = createScene();

  // run the render Loop
  engine.runRenderLoop(function(){
    scene.render();
  });

  // the canvas/window resize event handler
  window.addEventListener('resize', function(){
    engine.resize();
  });
});
</script>
</body>
</html>

```



Glitch.com

We'll be using Glitch.com as it provides an online editor and lets us share our work. You don't need to create an account, but it would probably help.

There's a basic project based on the code above set up at <https://glitch.com/~babylon-intro-mar19-a>

The code is split into three parts (index.html, style.css and script.js), because this is the way glitch.com organises things.

index.html

```

<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html" charset="utf-8"/>
    <title>Babylon - Getting Started</title>

    <!-- import the webpage's stylesheet -->
    <link rel="stylesheet" href="/style.css">

    <!-- import Babylon.js -->
    <script src="https://cdn.babylonjs.com/babylon.js"></script>

    <!-- import the webpage's javascript file -->
    <script src="/script.js" defer></script>
  </head>
  <body>
    <canvas id="renderCanvas"></canvas>
  </body>
</html>

```

```
</body>
</html>
```

style.css

```
html, body {
  overflow: hidden;
  width : 100%;
  height : 100%;
  margin : 0;
  padding : 0;
}

#renderCanvas {
  width : 100%;
  height : 100%;
  touch-action: none;
}
```

script.js

```
window.addEventListener('DOMContentLoaded', function(){
  var canvas = document.getElementById('renderCanvas');

  var engine = new BABYLON.Engine(canvas, true);

  var createScene = function(){
    var scene = new BABYLON.Scene(engine);

    var camera = new BABYLON.FreeCamera('camera1', new BABYLON.Vector3(0, 5,-10), scene);
    camera.setTarget(BABYLON.Vector3.Zero());
    camera.attachControl(canvas, false);

    var light = new BABYLON.HemisphericLight('light1', new BABYLON.Vector3(0,1,0), scene);

    var sphere = BABYLON.Mesh.CreateSphere('sphere1', 16, 2, scene);
    sphere.position.y = 1;

    var ground = BABYLON.Mesh.CreateGround('ground1', 6, 6, 2, scene);

    return scene;
  }

  var scene = createScene();

  engine.runRenderLoop(function(){
    scene.render();
  });

  window.addEventListener('resize', function(){
    engine.resize();
  });
});
```

Shapes

https://doc.babylonjs.com/how_to/set_shapes

```
var box = BABYLON.MeshBuilder.CreateBox("box", {height: 5}, scene);
var sphere = BABYLON.MeshBuilder.CreateSphere("sphere", {diameter: 2, diameterX: 3}, scene);
var cone = BABYLON.MeshBuilder.CreateCylinder("cone", {diameterTop: 0, tessellation: 4}, scene);
var plane = BABYLON.MeshBuilder.CreatePlane("plane", {width: 5}, scene);
var ground = BABYLON.MeshBuilder.CreateGround("gd", {width: 6, subdivisions: 4}, scene);
var disc = BABYLON.MeshBuilder.CreateDisc("disc", {tessellation: 3}, scene);
var torusKnot = BABYLON.MeshBuilder.CreateTorus("torus", {thickness: 0.2}, scene); var torus =
BABYLON.MeshBuilder.CreateTorusKnot("tk", {}, scene);
var ground = BABYLON.MeshBuilder.CreateGroundFromHeightMap("gdhm", url, {width: 6, subdivisions: 4}, scene);
var tiledGround = BABYLON.MeshBuilder.CreateTiledGround("tgd", {subdivisions: {w:4, h:6} }, scene);
```

Mesh vs MeshBuilder

The newer MeshBuilder method favours named parameters in an options list.

e.g.

```
var sphere = BABYLON.Mesh.CreateSphere('sphere1', 16, 2, scene);
```

```
var sphere = BABYLON.MeshBuilder.CreateSphere("sphere", {diameter: 2, diameterX: 3}, scene);
```


Part 4. Resources

Note: All these are accessible from the [Babylon.js homepage](#).

Examples

<https://doc.babylonjs.com/examples/>

Playground

<https://playground.babylonjs.com/indexStable.html>

Create Your Own Shader

<https://cyos.babylonjs.com/>

Documentation

<https://doc.babylonjs.com/>

Forum

<https://forum.babylonjs.com/>

Source

<https://github.com/BabylonJS/Babylon.js>

Sandbox

<https://sandbox.babylonjs.com/>

Part 5. It's Editing Time!

Moar Preparation (Sorry)

We still need to make a few more changes before we're ready to get stuck in. You can skip to the next section if this looks like too much like busy work.

Remix!

Reminder: The basic project is at <https://glitch.com/~babylon-mar19-b>

Click on the Remix to Edit button in the top right.



This will create your own version of the project to work on.

Take (Interactive) Control!

To the **JavaScript** replace with the camera code with the following:

```
var camera = new BABYLON.ArcRotateCamera("Camera", 0, 0, 10, new BABYLON.Vector3(0, 0, 0), scene);
camera.setPosition(new BABYLON.Vector3(0, 0, 20));
camera.attachControl(canvas, true);
```

Now we can control the position and orientation of the box with the mouse.

Add a Pile of Boxes

We need something to work with. Let's add a "pile" of boxes!

Replace

```
var sphere = BABYLON.Mesh.CreateSphere('sphere1', 16, 2, scene);
sphere.position.y = 1;

var ground = BABYLON.Mesh.CreateGround('ground1', 6, 6, 2, scene);

with

var shapes = [];
var numBoxes = 9;

for (let x = -numBoxes; x <= numBoxes; x++) {
  for (let y = -numBoxes; y <= numBoxes; y++) {
    var mat = new BABYLON.StandardMaterial("whiteMat", scene);

    var shape = BABYLON.MeshBuilder.CreateBox("box", {}, scene);
    shape.material = mat;
    shape.material.diffuseColor = new BABYLON.Color3(0, 1, 0);
    shape.position.x = x * 1.1;
    shape.position.y = y * 1.1;

    shapes.push(shape);
  }
}
```

Add a function to update the boxes, with a step parameter to control the speed of any changes we make. We need a way to make changes over time, a simple counter/step will suffice. This is just a skeleton function and doesn't anything just yet.

```
let step = 0;
function updateShapes(step) {
  for (let i = 0; i < shapes.length; i++) {

  }
}

engine.runRenderLoop(function(){
  step += 0.05;
  updateShapes (step);

  scene.render();
});
```

Functions

To produce interesting effects, maths functions are very useful, particularly: cos and sin. Since we'll be using them a lot let's alias the Math functions, we'll be using.

```
var cos = Math.cos;
var sin = Math.sin;
var sqrt = Math.sqrt;
var abs = Math.abs;
var PI = Math.PI;
```

Part 6. Animation

Messing about with values.

To get something interesting to happen on screen there are a number of properties we can influence:

Position, Scale, Rotation, Color*

* Please forgive the Americanism, I find it avoids confusion later on.

Position

```
function setShapePosZ(shape, step) {
  let x = shape.position.x;
  let y = shape.position.y;
  let md = abs(x) + abs(y);
  let d = sqrt(x*x + y*y);

  shape.position.z = x;
  shape.position.z = cos(x/2) + cos(y/2);
  shape.position.z = cos(d);
  shape.position.z = cos(d + step);
  shape.position.z = d;
}
```

Scale

```
function setShapeScaleZ(shape, step) {
  let x = shape.position.x;
  let y = shape.position.y;
  let md = abs(x) + abs(y);
  let d = sqrt(x*x + y*y);

  shape.scale.z = (d);
  //position.z = d;//cos(abs(x) - abs(y) + step);
}
```

Rotation

```
function setShapeRotZ(shape, step) {
  let x = shape.position.x;
  let y = shape.position.y;
  let md = abs(x) + abs(y);
  let d = sqrt(x*x + y*y);

  shape.rotation.z = ((step*5+d)*Math.PI/45);
  //position.z = d;//cos(abs(x) - abs(y) + step);
}
```

Color

```
function setShapeColor(shape, step) {
  let x = shape.position.x;
  let y = shape.position.y;
  let md = abs(x) + abs(y);
  let d = sqrt(x*x + y*y);

  shape.material.diffuseColor.r = cos(d+step);
}
```

Play time

Change shapes (sphere, torus, cylinder, cone).

We now have control of an array of attributes to create interesting effect.

We can now combine the different property changes, and tweak the functions for different effects.

Try changing to a positional spotlight, control its position.

Standard material has some nice additional attributes, such as shininess. Try changing them.

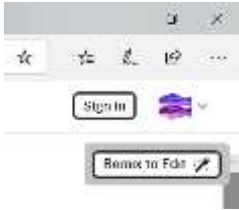
Part 7. Vertices

OK! We're finally ready to get something interesting going.

Remix Reminder!

There's a starter project setup at <https://glitch.com/~babylon-intro-mar19-d>

Click on the Remix to Edit button in the top right.



Fun Stuff or Something A Bit Different, SDF

Rather than playing around with whole objects, we can just as easily manipulate the individual vertices that make up an object.

```
var cos = Math.cos;
var sin = Math.sin;
var sqrt = Math.sqrt;
var abs = Math.abs;
var PI = Math.PI;

var ground;

window.addEventListener('DOMContentLoaded', function(){
  var canvas = document.getElementById('renderCanvas');

  var engine = new BABYLON.Engine(canvas, true);

  var createScene = function(){
    var scene = new BABYLON.Scene(engine);

    var camera = new BABYLON.ArcRotateCamera("Camera", 1, 1, 10, new BABYLON.Vector3(0, 0, 0), scene);
    camera.setTarget(BABYLON.Vector3.Zero());
    camera.attachControl(canvas, false);

    var light = new BABYLON.HemisphericLight('light1', new BABYLON.Vector3(0,1,0), scene);

    var mat = new BABYLON.StandardMaterial("whiteMat", scene);
    //mat.wireframe = true;
    mat.backFaceCulling = false;

    ground = BABYLON.MeshBuilder.CreateGround("gd", {width: 6, height: 6, subdivisions: 40, updatable:true},
scene);
    ground.material = mat;
    ground.rotation.x = 1;

    let positions = ground.getVerticesData(BABYLON.VertexBuffer.PositionKind);

    return scene;
  }

  var scene = createScene();

  let step = 0;

  engine.runRenderLoop(function(){
    step += 0.05;
    //console.log(ground);
```

```

    updateGround(ground, step);
    scene.render();
});

window.addEventListener('resize', function(){
    engine.resize();
});
});

function updateGround(ground, step) {
    //if(ground === undefined || ground === null) return;
    let positions = ground.getVerticesData(BABYLON.VertexBuffer.PositionKind);
    flo2(positions, step);
    ground.updateVerticesData(BABYLON.VertexBuffer.PositionKind, positions);
    var indices = ground.getIndices();

    var normals = [];

    //Calculations of normals added
    BABYLON.VertexData.ComputeNormals(positions, indices, normals);

    ground.updateVerticesData(BABYLON.VertexBuffer.NormalKind, normals);
}

function wave(vertices, step){
    var length = vertices.length/3;
    for (var i = 0; i < length; i++) {
        let x = vertices[i*3];
        let y = vertices[i*3+2];
        let r1 = Math.sqrt(x*x+y*y)*3;
        let r2 = (Math.abs(x)+Math.abs(y))*2;
        let r3 = Math.abs(Math.abs(x)-Math.abs(y));
        let rad = Math.atan2(y, x)
        //vertices[i*3+1] = (Math.cos(r1*r2+step)/30);
        vertices[i*3+1] = (Math.cos(r3));
    }
}

function sqwave(vertices, step){
    var length = vertices.length/3;
    for (var i = 0; i < length; i++) {
        let x = vertices[i*3];
        let y = vertices[i*3+2];
        let r1 = Math.sqrt(x*x+y*y)*3;
        let r2 = (Math.abs(x)+Math.abs(y));
        let r3 = Math.abs(Math.abs(x)-Math.abs(y));
        let rad = Math.atan2(y, x)
        let b = Math.abs(x)<Math.abs(y)?Math.abs(x):Math.abs(y);
        //vertices[i*3+1] = (Math.cos(r1*r2+step)/30);
        //vertices[i*3+1] = Math.ceil(Math.cos(r1+step/20));
        vertices[i*3+1] = Math.ceil(Math.cos(Math.ceil(r2)+step/2));
        //vertices[i*3+1] = Math.cos(b*1+step);
        //vertices[i*3+1] = Math.cos(r2+b+step)/2;
    }
}

function hemi(vertices, step){
    var length = vertices.length/3;
    for (var i = 0; i < length; i++) {
        let x = vertices[i*3];
        let y = vertices[i*3+2];
        let r = Math.sqrt(x*x+y*y); //Math.abs(x) + Math.abs(y); //
        // r += Math.sqrt(x*x+y*y);
        x=1-(r+1)%2;
        r = Math.sqrt(1-x*x);
        vertices[i*3+1] = r; //Math.cos(r*2)/5;
        //vertices[i*3+1] += Math.sin(r*5)/5;
    }
}

```

```

}
}

function manhat(vertices, step){
  var length = vertices.length/3;
  for (var i = 0; i < length; i++) {
    let x = vertices[i*3];
    let y = vertices[i*3+2];
    let r1 = (Math.abs(x)+Math.abs(y));
    let r2 = ( Math.abs(x) + Math.abs(y) );
    let r3 = Math.abs(x)>Math.abs(y)?Math.abs(x):Math.abs(y);
    let r = Math.sqrt(x*x+y*y);
    if(x<0 && y<0)
      vertices[i*3+1] = Math.cos(r+step);
    else
      vertices[i*3+1] = Math.cos(r3+step);
    //vertices[i*3+1] = Math.cos(r3+step);
  }
}

function mix(vertices, step){
  var length = vertices.length/3;
  for (var i = 0; i < length; i++) {
    let x = vertices[i*3];
    let y = vertices[i*3+2];
    let r1 = (Math.abs(x)+Math.abs(y));
    let r2 = x+y;
    let r3 = Math.max(Math.abs(x),Math.abs(y));
    let r = Math.sqrt(x*x+y*y);

    vertices[i*3+1] = Math.cos(-r1+r3-r+step);
  }
}

function heart(vertices, step){
  var length = vertices.length/3;
  for (var i = 0; i < length; i++) {
    let x = Math.abs(vertices[i*3]);
    let y = Math.abs(vertices[i*3+2]);
    let r1 = Math.pow((x*x + y*y - 1), 3) - x*x*x*y*y*y;
    let r2 = (x*x*x*x+y*y*y*y) - 2*x*x*y;
    let r3 = Math.max(Math.abs(x),Math.abs(y));
    let r = Math.sqrt(Math.abs(r2));

    let d = Math.pow((x*x+ y*y), 2) + 4*Math.abs(x)*(x*x+y*y)-4*y*y;

    vertices[i*3+1] = Math.cos(r)/2;
  }
}

function flo2(vertices, step){
  var length = vertices.length/3;
  for (var i = 0; i < length; i++) {
    let x = vertices[i*3];
    let y = vertices[i*3+2];
    let r = Math.sqrt(x*x+y*y);
    let m = (Math.abs(x)+Math.abs(y));
    let b = Math.max(Math.abs(x),Math.abs(y))*0.707;

    vertices[i*3+1] = Math.cos(2.5*(m+b-r+step/2))/2;
  }
}

function flo(vertices, step){
  var length = vertices.length/3;

```



```
for (var i = 0; i < length; i++) {  
  let x = vertices[i*3];  
  let y = vertices[i*3+2];  
  let r = Math.sqrt(x*x+y*y);  
  let m = (Math.abs(x)+Math.abs(y));  
  let b = Math.max(Math.abs(x),Math.abs(y))*0.707;  
  
  vertices[i*3+1] = Math.cos(2.5*(b-r+step/4))/2;  
}
```

Part 8. More advanced stuff

Wing it?