WebGL Workflow: Clara.io & Babylon.js

WebGL is great, but what sort of workflow is used to get from concept to a final element on a webpage? Here we examine one possible workflow.

Our workflow outline:

1. Initial concept
2. Create or acquire 3D assets
3. Customise or modify
4. Incorporate the model or scene into a web page

Concept: Create and Configure 3D Card Suits

I've always been fascinated by the various symbols used in games and thought they would be cool to use in a video game or just as decoration.

We'll use Clara.io (because I say so) to build a couple of the suit pips from playing cards (hearts ♥ , clubs ♣, diamonds ♦ and spades ♠) before incorporating them into a web page using Babylon.js. Along the way, we'll take a look at some of the features of Clara.io and Babylon.js.

A couple of possible approaches spring to mind:

1. Loft or extrude a polygon


2. Full on 3D model


We'll go for option 2.

Clara.io

1. Navigate to Clara.io

You should already have registered for a free account and be logged in (do so now if you haven't already… tsk).

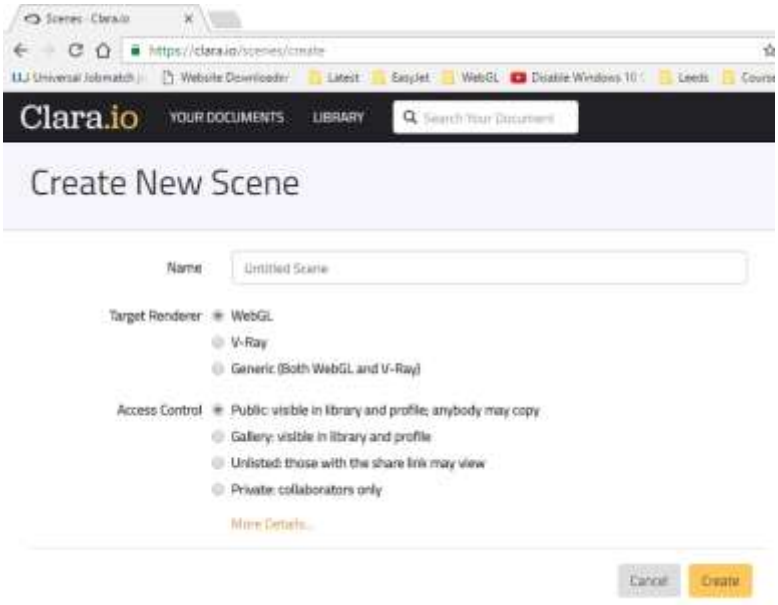
1.1. Create a New Scene

1. Click [+New Scene] or "Create a New Scene".

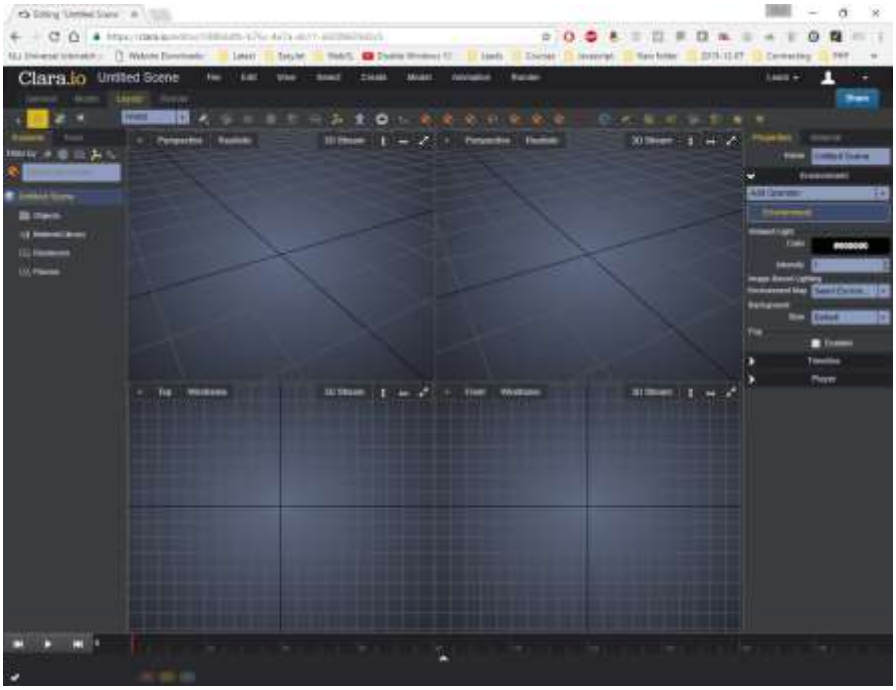
2. Click [Create Empty Scene].

(Note that there a number of pre-made scenes you can use as a base for your project.)

3. Enter a meaningful name, leave the other settings at their defaults and click [Create].



You should see something like this.



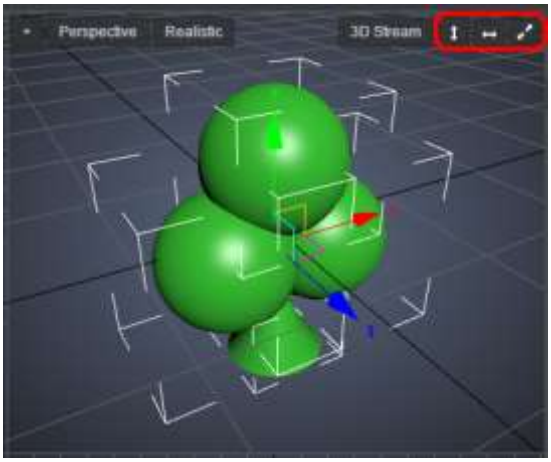
1.2. Full screen (F11)

Maximise screen real estate, just to make life that little bit easier

1.3. Basic controls and navigation

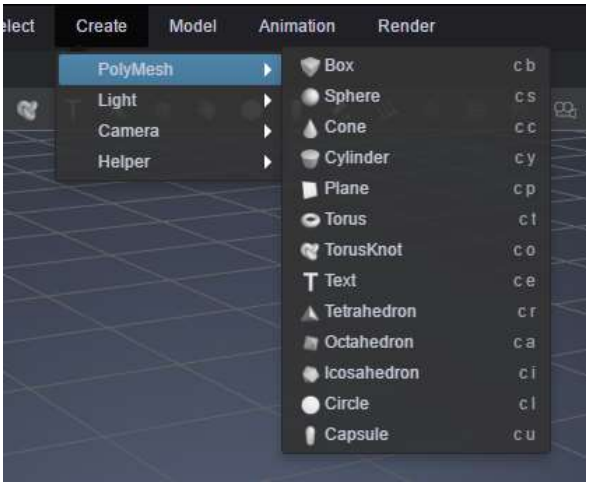
There's a lot of info on the Clara.io site, but we'll quickly cover navigation and selection here.

1. To maximise a viewport click on the arrow in the top right of the viewport.



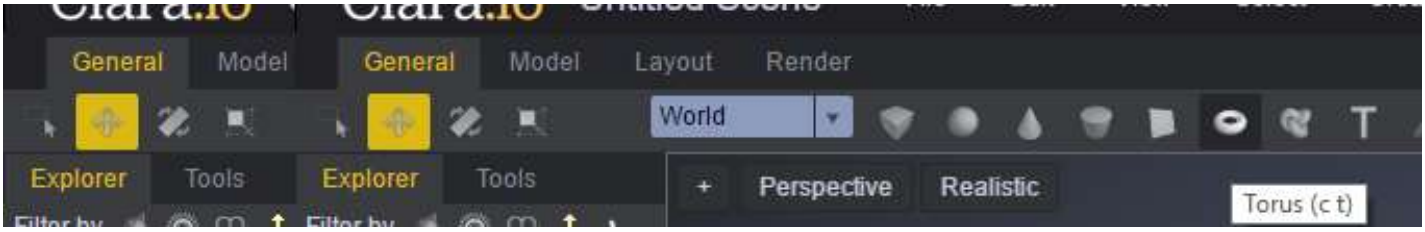
2. Add a few models (create a few polymeshes).

Either [Create] → [PolyMesh] → PolyMesh

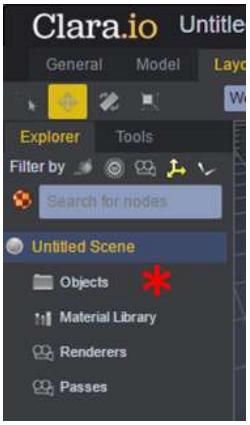


OR

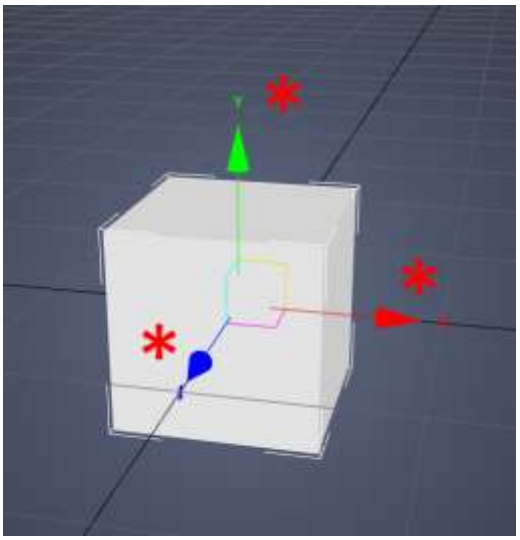
Select from the shapes along the top of the viewport.



Select an item by clicking on it, either on screen or the explorer list on the left.



Move items by selecting the axis or plane manipulator gizmo and dragging.



Add items to the **selection** by holding the "**Ctrl**" key and clicking on them.

Remove items from the **selection** by holding the "**Alt**" key and clicking on them.

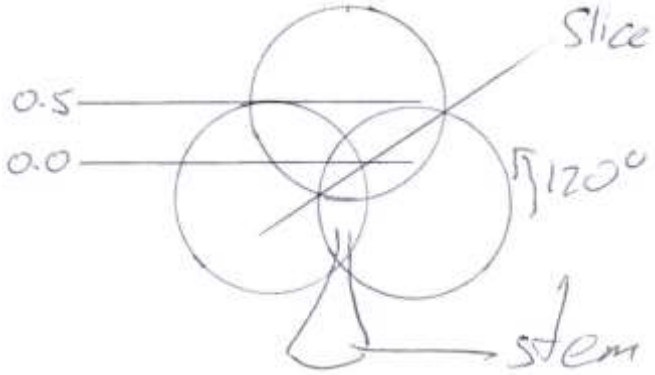
Press the "F" key on the keyboard to focus on selected items.

Mac	Linux
• Pan - Command + Ctrl + Left Click • Orbit - Command + Left Click • Zoom - Scroll (two finger drag)	• Pan - Ctrl + Middle Click • Orbit - Middle Click • Zoom - Scroll (Mouse Wheel)

Windows - three button mouse	Windows - two button mouse
• Pan - Ctrl + Middle Click • Orbit - Middle Click • Zoom - Scroll (Mouse Wheel)	• Pan - Ctrl + Alt + Right Click • Orbit - Alt + Right Click • Zoom - Ctrl + Alt + Left Click

2. Create “clubs” model

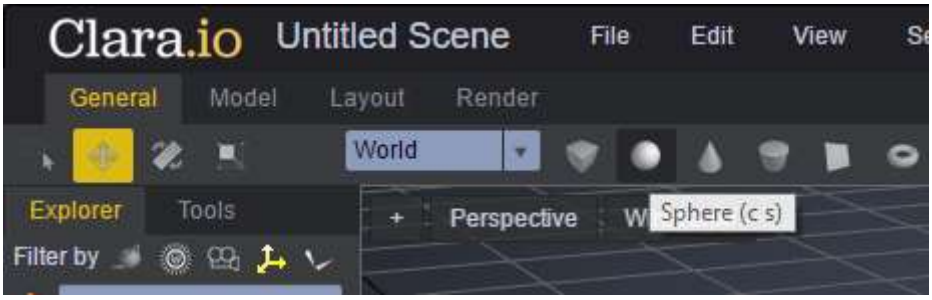
A (very) rough diagram to give an idea of the components (based on a unit sphere), their positions and orientations making up the club.



Since I don’t like having a lot of unseen vertices (where the spheres overlap), we’re going to trim them away. This isn’t just whimsy on my part, the fewer vertices there are the quicker rendering will be and the less memory used.

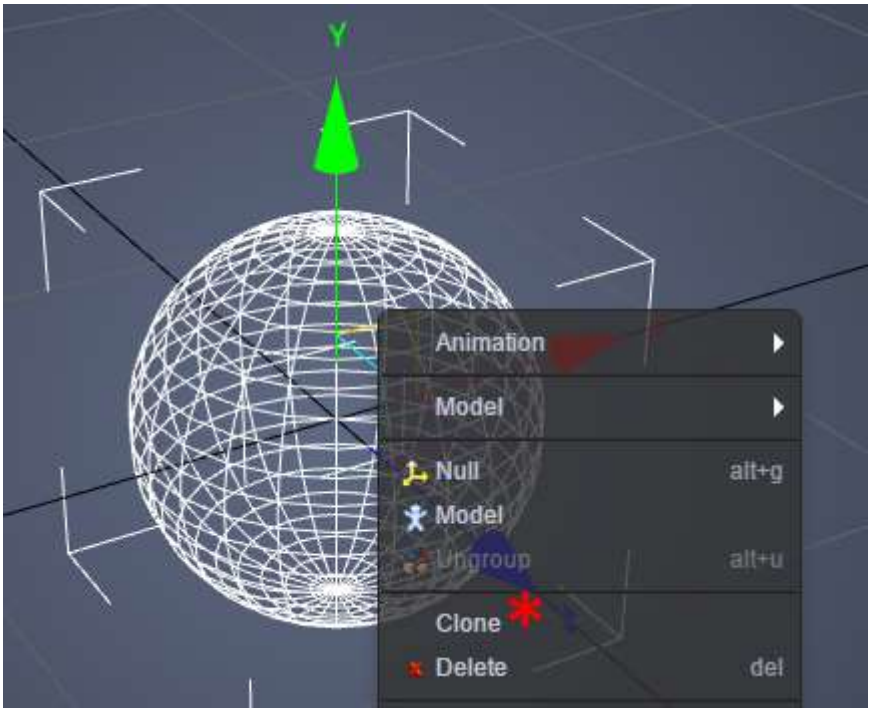
1. Switch to Wireframe view.

2. Add a sphere to the scene.

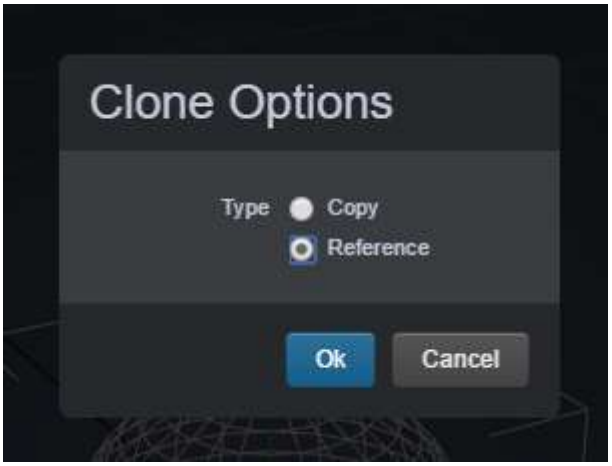


3. **Clone** the sphere.

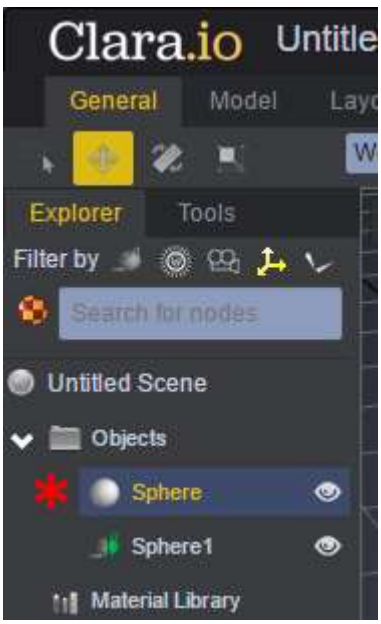
Right-click the sphere, then select Clone



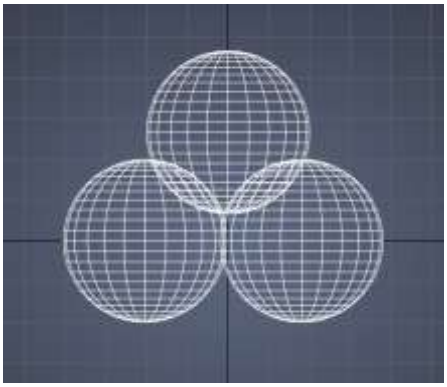
Ensure Reference is checked, then click OK.



4. Repeat step 2 (make sure to select the original sphere and not the exisitng clone – move the two spheres apart or select it from the object list on the left).

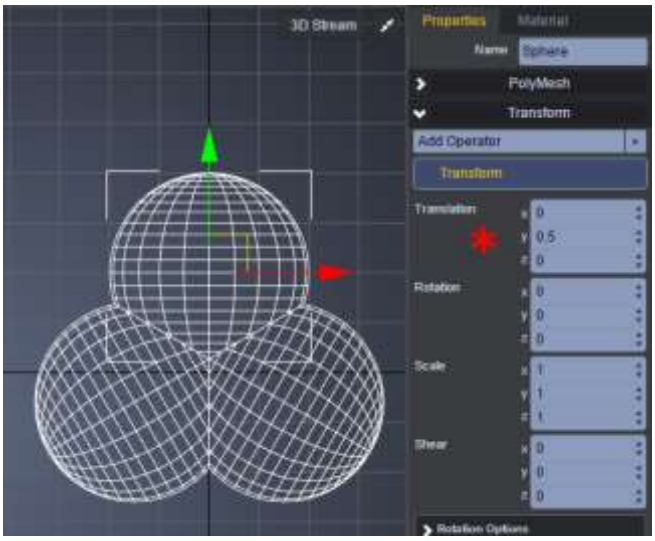


5. Move the spheres to these approximate positions (the original is in the middle and raised).



6. Tidy up the positions of the spheres, so that they overlap by the same amount. We do this by entering exact figures in their property tranforms.

Select the top sphere, then in the transform section / translation enter 0.5 in the y field.

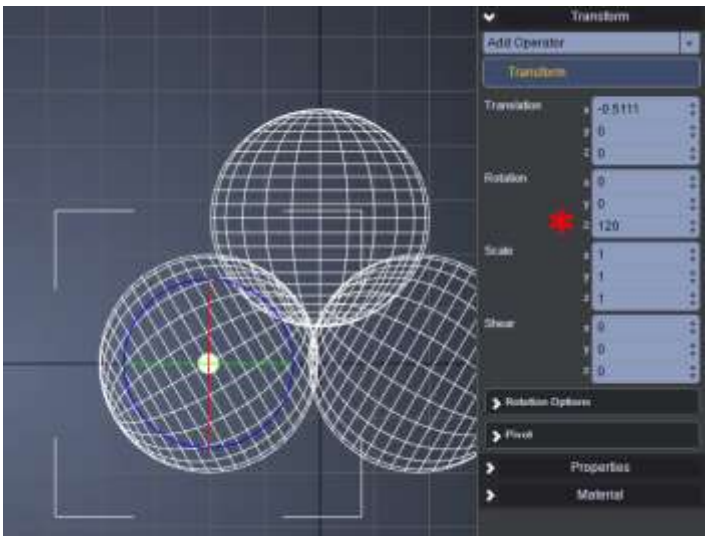


For the left sphere set x to 0.375 and y to -0.15.

For the right set x to -0.375 and y to -0.15.

7. Rotate the left clone by 120 degrees about the z-axis.

Select the left clone, then in the transform section / rotation enter 120 in the z field.



8. Repeat for the right clone, but rotate by -120.

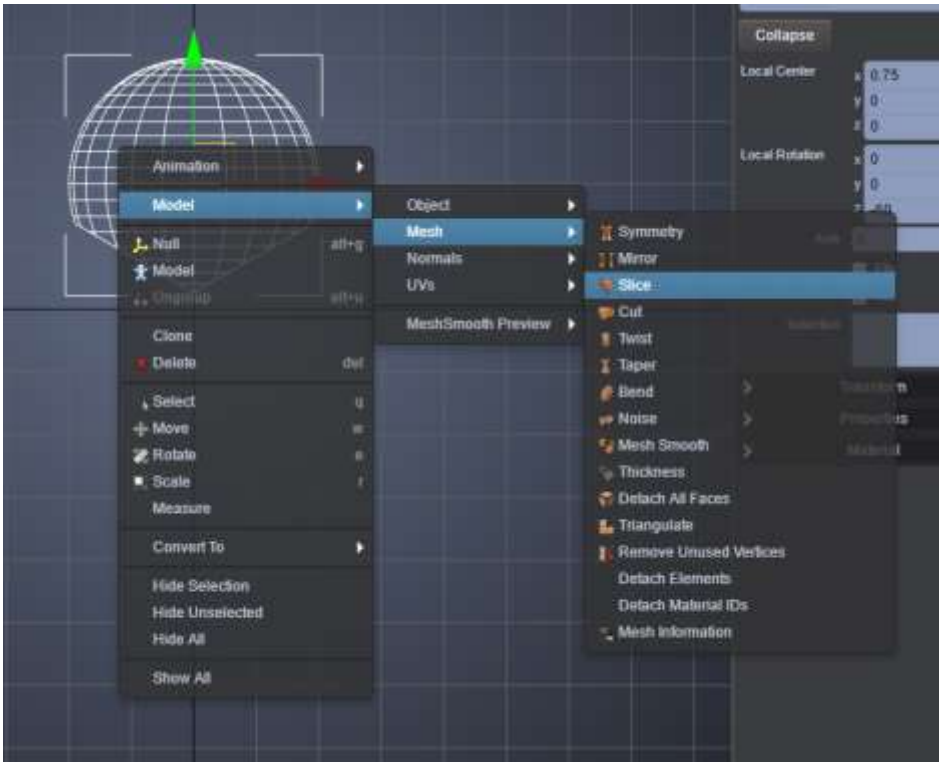
Why all this fiddling with the rotation? Well, to trim the excess vertices we could use the Boolean operator (Model → Model (Experimental)) but the function is still experimental and the result isn't very clean.

Instead we'll slice the excess vertices, and because we cloned the latter two spheres we only need to do this to the parent sphere and the changes will be inherited.

Effectively we end up with three sliced spheres which we'll piece together.

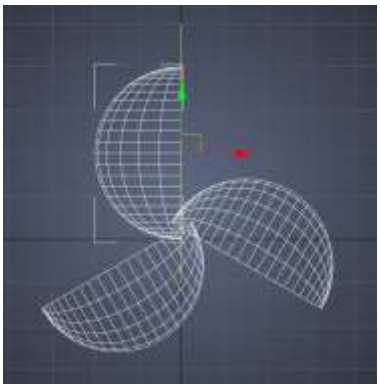
9. Slice excess vertices from sphere.

Right-click the visible sphere, then Model→Mesh→Slice



A Slice operator will be added to the PolyMesh properties.

You should see something like this, with the two clones inheriting the slice from the parent..



(Placing the slice correctly took a bit of trail and error.)

Adjust the slice by selecting it from the Properties tab (on the right) → PolyMesh → Slice/Cut

Set Local Centre x to -0.75

Set Local Rotation z to -120.

10.Add a second slice by repeating step 9.

Set Local Centre x to 0.75

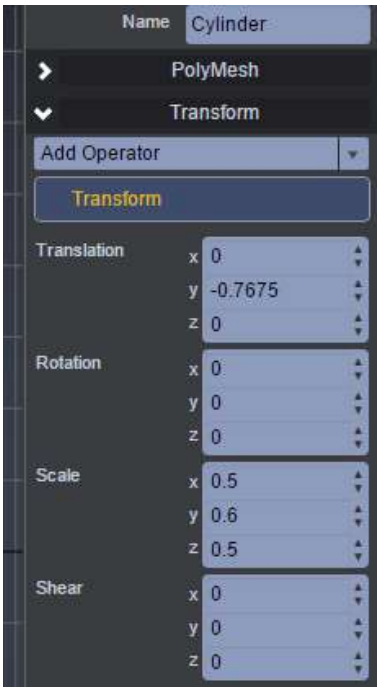
Set Local Rotation z to 60.

11.Add a stem.

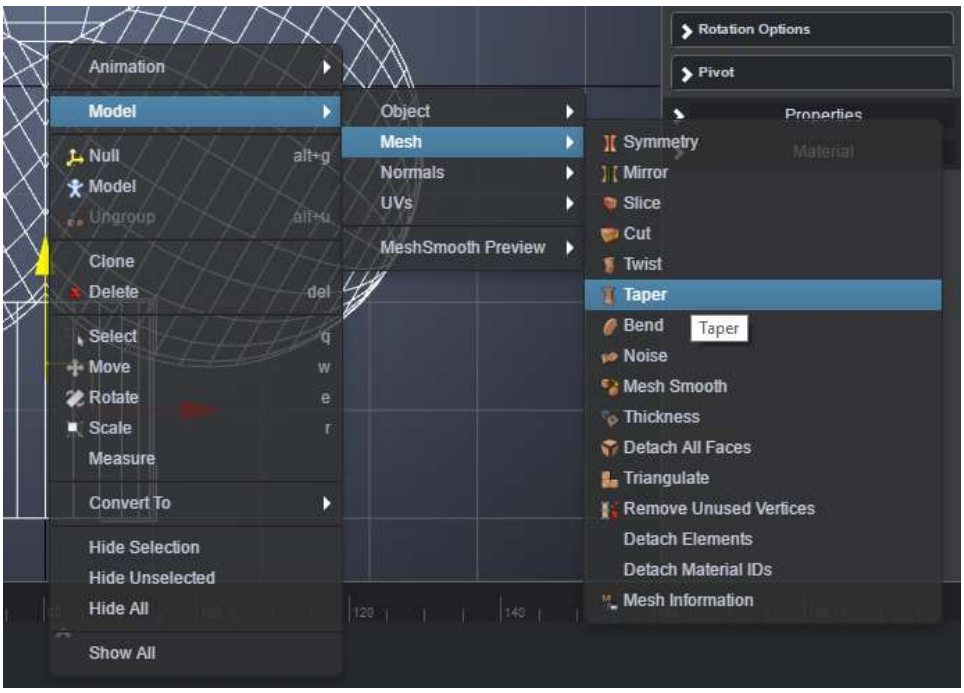
Add a cylinder.

Place it just below the spheres.

Set its transform and properties.



Add a taper to it.



There's no apparent effect, we need to update the cylinders properties.

OK, OK. There are some hidden vertices in the stem. Let's just move on, shall we?

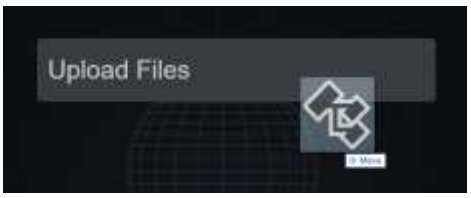
3. Create "hearts" model

3.0. An aside

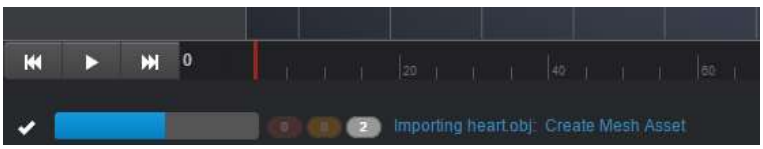
Bit of a cheat here, there are several formulae that describe heart shaped geometry (called cartoid functions) and having implemented one in Three.js, it seems a waste not to use it. So I used the `OBJExporter` with a bit of copy and paste to create an .obj file. The source files are included for completeness.

3.1. Load model

Drag and drop the heart.obj file onto the Clara.io workspace.



While loading you see something like this in the bottom left.



The club and background grid have disappeared, that's because the heart is B-I-G!

3.2. Edit model

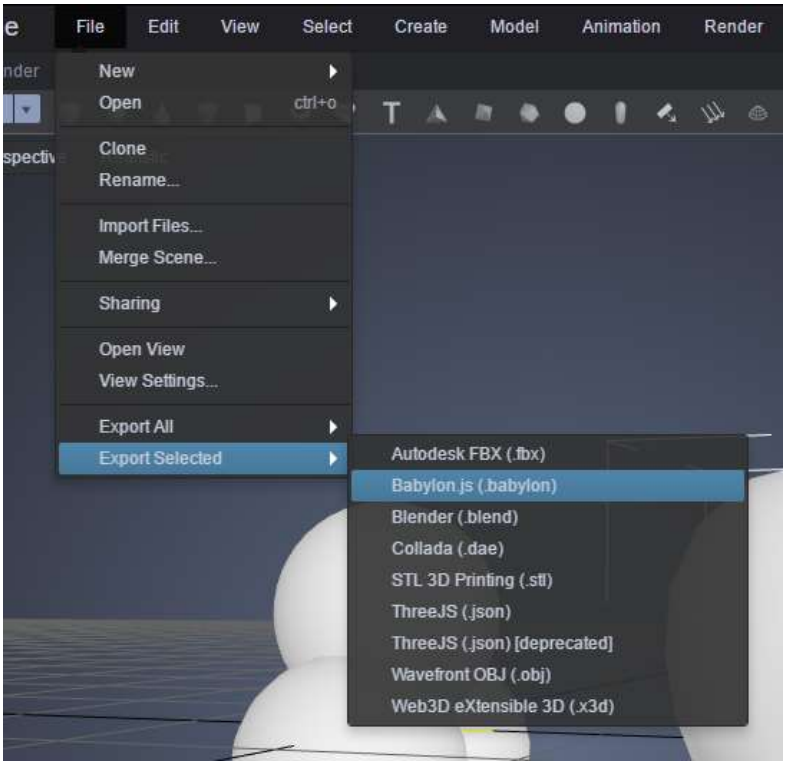
Scale down the model by 0.01 in each dimension.

4. Create “clubs” and “spades” models

These are left as exercises for the student.

5. Export the models

Select the heart,



Repeat for the club, but select all the component parts

Displaying in Babylon

Below is the code for a basic web page to create a Babylon scene and import the two models we've created.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html" charset="utf-8" />
  <title>Clara.io and Babylon.js</title>
  <script src="babylon.js"></script>
  <script src="dat.gui.min.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 () {
      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("ArcRotateCamera", 1, 0.8, 10, new BABYLON.Vector3(0, 0,
0), scene);

        BABYLON.SceneLoader.ImportMesh("", "", "club2.obj", scene, function (newMeshes) {
          newMeshes[0].position.x -= 2;
        });

        BABYLON.SceneLoader.ImportMesh("", "", "hearts.babylon", scene, function (newMeshes) {
          console.log(newMeshes);
          newMeshes[0].scaling.x = 0.01;
          newMeshes[0].scaling.y = 0.01;
          newMeshes[0].scaling.z = 0.01;

          newMeshes[0].position.x = 1;

          newMeshes[0].rotation.y = Math.PI / 2;

        });

        camera.setTarget(BABYLON.Vector3.Zero());

        camera.attachControl(canvas, false);

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

        return scene;
      }

      var scene = createScene();

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

```
      });

      window.addEventListener('resize', function () {
        engine.resize();
      });
    });
  </script>
</body>
</html>
```

Some points to note, the two have greatly different scales.