Web 3D Formats

**Table of Contents**

1 Introduction 2

2 Testing your model in threejs editor. 4

3 Setting up a 3D scene in your HTML 6

4 Javascript for 3D Cube 8

5 Adding the Coke Can model to a web page. 12

6 Adding a card to display your 3D animation. 17

# Introduction

Lab 3 has been designed to introduce you to the process of converting your 3D models from Blender into a Web 3D format, that will allow you to display your 3D model and interact with it on the Internet.

There are several ways to allow 3D to be rendered in your 3D App, but we will largely focus on exploiting a JavaScript technology.

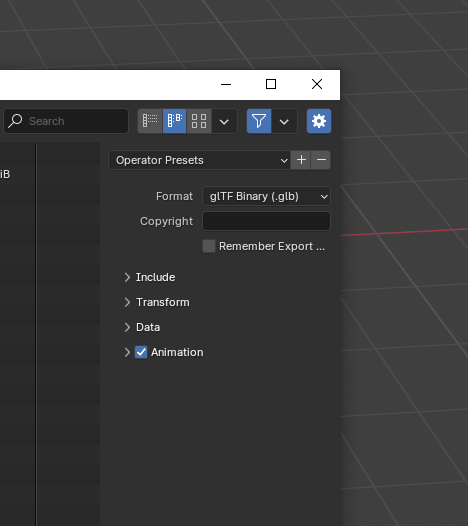
By scene, we mean your 3ds Max, C4D, Blender or Maya 3D models converted to a model to display. Initially we will export your .blend file into a GLB (The GLB locates all of the elements part of a 3D scene, including materials, node hierarchy and cameras, in one single compressed file).

In Blender, go to file>export and select gLTF option (GL Transmission Format and formerly known as WebGL Transmissions Format or WebGL TF is a standard file format for three-dimensional scenes and models).

Make sure that you are exporting the glTF Binary option which is .glb single file. The other option does allow you separate textures. Make sure that Animation is ticked. Your exported file needs to be saved into your 3D App folder from Lab 1.

A screenshot of a computer

Description automatically generated



# Testing your model in threejs editor.

You can test your exported .glb model in the threeJS editor or use this tool to create 3D models and assets. Just open the editor > go to file and import. You will need to add a light source to see your model and this is a great way of working out which lights are best in your scene when you come to write the Javascript.

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A computer screen shot of a soda can

Description automatically generated

# Setting up a 3D scene in your HTML

We will initially focus on how to apply ThreeJS to your webpages from Lab 1. Here we will add a rotating cube to your index.html just to give you a bit of practice using Javascript. To make this work your Javascript needs to:

Load Thre.JS

Create a Scene

Create a Camera

Create a Renderer – webGL renderer

Create an object which is a mesh: A mesh has two perametres. One; it has geometry vertices, faces and UVs and two it has a material.

We will also load a light.

A lot of the script used in these tutorials comes direct from the ThreeJS website: <https://threejs.org/>

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

To add the Javascript – add the following into your index.html:

|  |
| --- |
| **<script src='https://cdnjs.cloudflare.com/ajax/libs/three.js/106/three.min.js'></script>**  **<script src='https://s3-us-west-2.amazonaws.com/s.cdpn.io/2666677/OrbitControls.js'></script>**  **<script src='https://s3-us-west-2.amazonaws.com/s.cdpn.io/2666677/GLTFLoader.js'></script>**  **<script  src="./script.js"></script>** |

These lines include external JavaScript libraries that are typically used for 3D rendering and interactive web-based 3D content.

**Three.js (three.min.js):**

This is the core library used to create and render 3D graphics in the web browser using WebGL. This is the primary engine for rendering 3D content on the web.

**OrbitControls.js:**

This is an additional utility script that enhances the functionality of Three.js by allowing interactive control over the 3D scene. It enables features like rotating, zooming, and panning the camera around a 3D scene using mouse movements.

**GLTFLoader.js:**

This script is used to load 3D models in the GLTF (GL Transmission Format) file format, which is a modern, efficient, and widely-supported format for 3D assets. By including this script, you can load 3D models directly into a Three.js scene from external sources in GLTF or GLB formats. We will need this to load your exported .blend model into the web page.

The final line (<script  src="./script.js"></script>) is need to load the Javascript you are going to write. In Visual Studio Code, right-click in the explorer and add a new file. Remember to call this script.js.

A screenshot of a computer

Description automatically generated

# Javascript for 3D Cube

Below is the information to add into your script.js file to add the cube into your index.html and what each line is doing.

|  |  |
| --- | --- |
| var scene, camera, renderer, box;  init();  function init(){    scene = new THREE.Scene();    scene.background = new THREE.Color(0xaaaaaa);    camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 0.1, 1000 );  camera.position.z = 3;  renderer = new THREE.WebGLRenderer();  renderer.setSize( window.innerWidth, window.innerHeight );  document.body.appendChild( renderer.domElement );    const light = new THREE.DirectionalLight();  light.position.set(0, 1, 2);  scene.add(light);    const material = new THREE.MeshStandardMaterial({color: new THREE.Color('skyblue')});    const geometry = new THREE.BoxGeometry(1, 1, 1);  box = new THREE.Mesh(geometry, material);  box.position.x = 0;  scene.add(box);  window.addEventListener('resize', onResize, false);    update();  }  function update(){  requestAnimationFrame(update);    box.rotation.y += 0.01;    // Render the scene  renderer.render(scene, camera);  }  function onResize(){  camera.aspect = window.innerWidth / window.innerHeight;  camera.updateProjectionMatrix();  renderer.setSize(window.innerWidth, window.innerHeight);  } | Global variables.  Default colour is black  Three.js initialization code  Container which holds the objects we want to render.  Colour background  Camera (perspective camera so things look smaller the further away they are). 4 parameters [FOV field of view. Degrees. Aspect ratio – width of the screen by height, near and far values].  Renderer to render on to the screen.  HTML5 canvas component.  Light in the scene. Directional light simulates light coming from a specific direction, like sunlight.  This sets the position of the light in 3D space.  To colour our cube. (Standard material is more complex and requires light)  Building a 3D object. Geometry of a cube  Adds to the scene.  Browser is calling the update about 60 frames per second.  If the window resizes, this changes the view of the object. |

A screenshot of a computer screen

Description automatically generated

View the result using the live server extension.

A blue square with a grey background

Description automatically generated

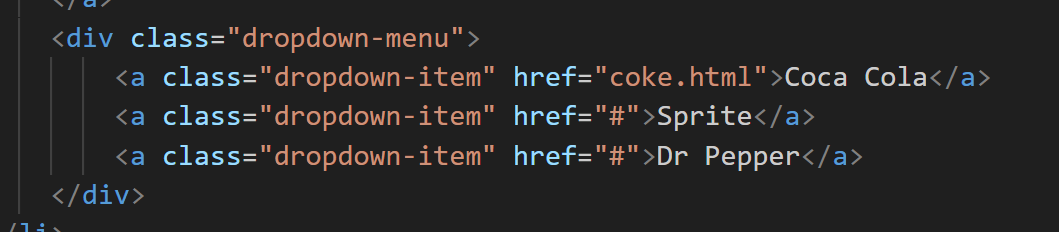
A blue cube on a grey background

Description automatically generated

# Adding the Coke Can model to a web page.

Now we have something working on our web page lets add our can model from lab 2. Duplicate your index.html and rename this coke.html. This new html file is going to be our coke can example.

You can go through and exchange the ‘#’ placeholders we put in so that your index page links to your coke.html and your coke.html links back.



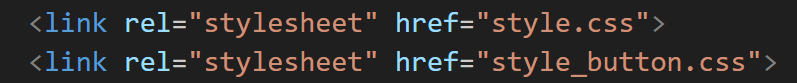
You will need to copy across the all of the Javascript links from the cube above into your coke.html. You should delete the sections of your Bootstrap grid.

We are now going to create a button to play the animation we created in Blender. In your coke.html add an a tag linking to button.

A black background with white text

Description automatically generated

We are going to add some design to this. So in your coke.html add a link to a new .css file and create the file in same way we created the Javascript. Right-click and name style\_button.css



A screen shot of a computer

Description automatically generated

Click on style\_button.css to open in a tab in Visual Studio Code. Add the following design. We are going to amend the information and you can change the colours and fonts now if you want to.

A screen shot of a computer program

Description automatically generated

The script below allows you add your export .GLB to your web page. It is important to experiment with the settings to see what they are doing. Make sure you direct the path to where your model is stored. In this code we are going to add an animation. This code initializes an AnimationMixer for a model and loops through the animations in a GLTF file. It converts each animation clip into an action that can be controlled by the AnimationMixer, and stores these actions in an actions array for future use (such as playing, pausing, or blending animations).

|  |  |
| --- | --- |
| var scene, camera, renderer, clock, mixer, actions = [], mode;  init();  function init() {  const assetPath = './'; // Path to assets      clock = new THREE.Clock();      // Create the scene  scene = new THREE.Scene();  scene.background = new THREE.Color(0x00aaff);    // Set up the camera  camera = new THREE.PerspectiveCamera(60, window.innerWidth / window.innerHeight, 0.1, 1000);  camera.position.set(-5, 25, 20);      // Add lighting  const ambient = new THREE.HemisphereLight(0xffffbb, 0x080820, 1);  scene.add(ambient);    const light = new THREE.DirectionalLight(0xFFFFFF, 2);  light.position.set(0, 10, 2);  scene.add(light);    // Set up the renderer  renderer = new THREE.WebGLRenderer();  renderer.setSize(window.innerWidth, window.innerHeight);  document.body.appendChild(renderer.domElement);  // Add OrbitControls  const controls = new THREE.OrbitControls(camera, renderer.domElement);  controls.target.set(1, 2, 0);  controls.update();    // Button to control animations  mode = 'open';  const btn = document.getElementById("btn");  btn.addEventListener('click', function() {  if (actions.length === 2) {  if (mode === "open") {  actions.forEach(action => {  action.timeScale = 1;  action.reset();  action.play();  });  }  }  });    // Load the glTF model  const loader = new THREE.GLTFLoader();  loader.load(assetPath + 'can\_template\_animation\_tab.glb', function(gltf) {  const model = gltf.scene;  scene.add(model);    // Set up animations  mixer = new THREE.AnimationMixer(model);  const animations = gltf.animations;  animations.forEach(clip => {  const action = mixer.clipAction(clip);  actions.push(action);  });  });      // Handle resizing  window.addEventListener('resize', resize, false);    // Start the animation loop  animate();  }  function animate() {  requestAnimationFrame(animate);  // Update animations  if (mixer) {  mixer.update(clock.getDelta());  }  renderer.render(scene, camera);  }  function resize() {  camera.aspect = window.innerWidth / window.innerHeight;  camera.updateProjectionMatrix();  renderer.setSize(window.innerWidth, window.innerHeight);  } | Add variables to your script.  If the is not correct then the script will not find your model.  The clock method helps time your animations.  This line of code positions the camera at the coordinates (-5, 25, 20), meaning the camera is:  5 units left of the origin (negative X),  25 units above the origin (positive Y),  20 units in front of the origin (positive Z).  Orbit controls to allow you to move your can around.  This links back to your “btn” in your html and css. The primary purpose of this is to control and reset animations, stored in the actions array, when the button with ID "btn" is clicked.  The model is our 3D .glb export file from Blender.  ThreeJS uses 4 stages to create animations:  1. Mixer  2. Clip  3. Action  4. Update  THREE.AnimationMixer is a class provided by THREE.js that is used to control the playback of animations in a 3D object.  The mixer object allows you to manage and control multiple animations that are applied to the model.  The forEach loop goes through each clip in the animations array.  For each clip (which represents an animation), the mixer.clipAction(clip) method is used to create an action that is associated with that particular animation clip.  The action represents an instance of the animation and is what you'll control (play, pause, stop, etc.) when managing the animation.  The created action is then added to the actions array using actions.push(action). |

Hopefully you now have a web page with your 3D model showing and when you click the button an animation plays. At the moment, it is stretched across the screen and the button might not be in the right place.

A red and black can on a blue background

Description automatically generated

# Adding a card to display your 3D animation.

We now need to crate a card to house our animation and display it. To achieve this we are going to add to the Javascript, Coke.html and the CSS we are using. Firstly, in the Javascript file find the renderer instruction. We need to add a canvas here so the animation is rendered into a space we can then control.

A computer screen shot of a computer code

Description automatically generated

In this code, we are getting the canvas element. The changes to this part of the code allows us to select a <canvas> element with the ID threeContainer and assign it to the variable canvas. Therefore, we are using the Three.js renderer, linking it to the canvas, and preparing it to render with optimal pixel ratio based on the device's display. We now need to adapt the resize function.

A screen shot of a computer program

Description automatically generated

By updating the function resize we will start to control how the model is displayed. To achieve this we need to get the canvas element and get the current width and height of the canvas. We then update the cameras aspect ratio based on the width and height and set the renderer to match the new width and height.

* Aspect Ratio: When the window is resized, the aspect ratio of the viewport changes. Updating the camera’s aspect ratio ensures the scene isn’t stretched or compressed inappropriately.
* Camera Update: Updating the camera’s projection matrix after changing the aspect ratio ensures the 3D camera continues to function properly with the new dimensions.
* Renderer Size: The renderer needs to match the viewport size to ensure that the 3D scene is rendered correctly. If you don’t adjust the renderer size, the output can look pixelated or incorrect after resizing.

**Coke.html**

The editing to the coke.html file controls how the canvas and container will appear. We have decided to put a title in as well as part of the look. Here we are adding the container and cards that will house our text and 3D models. I have also updated the bootstrap code to reflect that the open can is a button. I will add further buttons in the next lab.

A screen shot of a computer program

Description automatically generated

Below this script, I am adding the follow function so that the container resizes when the browser resizes.

A screen shot of a computer program

Description automatically generated

This code ensures that the height of #threeContainer will adjust dynamically whenever the browser window is resized, including when the window's height changes. The event listener makes sure the function on page loads when the window resizes.

**CSS**

In our CSS file we can add some more information to help the web page display as we intend. To start we need to add the following to the CSS.

A black screen with white text

Description automatically generated

This CSS code applies specific styles to elements with the classes .container-fluid, .row, and .col-sm-10. This set of rules helps with our layout by removing default spacing and ensure that elements stretch to fill their container without causing unwanted overflow or scrollbars.

Add the following text into your CSS.

A screen shot of a computer code

Description automatically generated

This CSS code applies to the threeContainer. Here we are controlling the view port height (vh).