Aleksi Pakkanen

RENDERING A 3D SCENE INSIDE A WEB APPLICATION

Bachelor’s thesis
Information Technology / Game Programming

2021

Kaakkois-Suomen
ammattikorkeakoulu

South-Eastern Finland
University of Applied Sciences

Tekijä/tekijät	**Tutkinto**	**Aika**
Aleksi Pakkanen	Insinööri (AMK)	Toukokuu 2021

Opinnäytetyön nimi 3D-näkymän renderöinti verkkoapplikaatiossa	33 sivua
Toimeksiantaja Kotkamills Oy	
Ohjaaja Marko Oras	

Tiivistelmä

Opinnäytetyön tehtävänä oli generoida kolmiulotteinen representaatio merikontista, jonka sisällä on useita paperirullia. Tämä tehtiin verkkoselaimessa toimivaan merikontin optimointityökaluun paremman visualisoinnin vuoksi. Tutkin tätä varten CSS 3D Transforms, WebGL 1.0- sekä JavaScript-kirjastoja Three.js sekä Babylon.js. Kolmiulotteinen kuva saatiin aikaiseksi olemassa olevaan työkaluun.

Three.js ladattiin omalle laitteelle GitHub-sivuilta ja sitten otettiin käyttöön verkkosovelluksessa. Eri muuttujat luotiin kameralle, näkymälle sekä renderöijälle, ja ne otettiin käyttöön Three.js-kirjaston komentoja käyttäen. PlaneGeometry-objekteja käytettiin merikontin 3D-mallin luomiseen. Sekä EdgeGeometry- että CylinderGeometry- muotoja käytettiin funktion luontiin, joka luo ja siirtää nämä paperirullat oikeisiin paikkoihin käyttäen optimointityökalun olemassa olevia funktioita. Kameran ohjaaminen annettiin käyttäjälle käyttäen Three.js OrbitControls-funktiota, ja renderöintisilmukka tehtiin, koska OrbitControlsin käyttö vaatii sellaista. Näkymän kaikkien Three.js-objektien muistista tyhjentämiseen tarvittiin myös funktio, sillä kaikki käyttäjän lisäämät paperirullat eivät välttämättä mahdu yhteen merikonttiin.

Projekti saatiin valmiiksi ja 3D-näkymän luonti käyttäen Three.js-kirjastoa saavutettiin. Three.js-kirjaston mainion dokumentaation vuoksi lähes jokainen ohjelmointiin liittyvä ongelma ratkaistiin helposti. Projekti koetaan onnistuneeksi, sillä työkalu on ollut käytössä toimeksiantajalla opinnäytetyön kirjoituksen aikana ja antanut siitä enimmäkseen positiivista palautetta.

Avainsanat

kolmiulotteisuus, verkko-ohjelmointi, tietokonegrafiikka

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree	Time
Aleksi Pakkanen	Bachelor of Engineer-ing	May 2021

Thesis title

Rendering a 3D scene in a web application

33 pages

Commissioned by

Kotkamills Oy

Supervisor

Marko Oras

Abstract

The objective of this thesis was to provide insight into the creation of 3D scenes inside web applications. For this purpose, a container optimization web application displaying a 3D representation of a container with varying numbers of paper reels inside of it was made. In order to support this, CSS 3D Transforms, WebGL and JavaScript 3D libraries Three.js and Babylon.js were studied, and the 3D scene was implemented into the existing web application.

In this study, Three.js was downloaded locally from its GitHub page and then loaded into the web application. Variables for the camera, scene and renderer objects were made and initialized using Three.js functions. PlaneGeometry objects were used in creating the 3D cargo container model. Both EdgeGeometry and CylinderGeometry shapes were used in creating a function that would place the paper reels in correct positions with the help of the application's placement optimization functions. Camera controls were given to the user using OrbitControls Three.js function, and a rendering loop was created because it was required for the OrbitControls functionality. Because all reels in a single order would not fit inside a single container, a function that would clear the scene from all Three.js objects were also created.

The creation of a suitable 3D scene was successfully achieved using Three.js. Because of its excellent documentation properties, most programming issues that arose during the study were easily solved. The commissioner successfully used the application giving mostly positive feedback.

Keywords

three-dimensionality, web programming, computer graphics

TABLE OF CONTENTS

Abbreviations

API – Application Programming Interface. An interface that allows the transmission of data between multiple separate applications.

CDN – Content Delivery Network. A group of interconnected servers across the globe that help speed up the delivery of internet content.

CSS – Cascading Style Sheets. A programming language used to stylize how HTML elements are displayed on screen.

DOM – Document Object Model. An API used in HTML and XML documents.

HTML – Hyper Text Markup Language. A programming language used in the creation of web pages.

PBR – Physically Based Rendering. A method of rendering which aims to match the reflections of light in the real world.

PDF – Portable Document Format. A file type used for electronic documents.

1 INTRODUCTION

Today, it seems reasonable to argue that web applications are the future of software development. Web applications are incredibly easy to distribute to countless users, and as the application runs on a browser software, little needs to be done to maintain support across various platforms. Modern technology has made it simple for web developers to create stunning 3D visuals inside web applications to enhance the user experience that functions regardless of the user's device.

1.1 Kotkamills Oy

Kotkamills Oy is a Finnish forest industry company located in Kotka, Finland, that has been producing paper and forestry products since 1872 (Kotkamills, n.d.). The company sends reels of paper all over the world and, thus, would gladly have a tool for automating the calculation of cargo containers needed for shipping their products and minimizing instances for making mistakes. Although many optimization tools for such purposes exist, they had proven to be slow or difficult to use, so the company requires a software tailored to tackle the problems they encounter more efficiently.

1.2 Purpose of this thesis

The purpose of this thesis is to examine the implementation of a 3D scene inside a HTML-file with the help of a few commonly available methods. The implementation of the 3D scene is made for a container optimization tool commissioned by Kotkamills and developed by two students.

This thesis studies four different ways of generating a 3D scene inside a web application. After analyzing each technology, a comparison is made, and the most suitable one is selected and used to solve the problem in the software provided by the commissioner. This problem will be described in greater detail in Chapter 1.3.

This thesis focuses on the 3D technology rather than the entire development of the container optimization tool. However, a brief overlook of the software in development is required to determine which solution should be chosen.

1.3 Container optimization tool

The container optimization tool is a browser-based application commissioned by Kotkamills Oy and developed by the author of this thesis and Peetu Seppälä. The software development started in June 2018 and finished in June 2019.

The software development was split between two programmers after the application's initial design phase, where a mockup image of the user interface was conducted by both programmers. The user interface and the 3D scene would be implemented by the author of this thesis, while the optimization math problem would be implemented by Peetu Seppälä. The additional features that would be added during development would be tasked in such a manner that all refining of the optimization would be developed by Peetu Seppälä, and everything else would be the author's responsibility. From now on in this thesis, this team of two programmers will be referred to as the development team.

1.4 Required features

When the project began, the list of features required in the container optimization application was quite short. The most important features comprised optimizing the calculation of a cylinder's position inside a container and showing this to the user in a 3D image. The number of required features increased during the application's development phase. The added features included a basic manipulation of the user interface as well as the ability to read data from a .xlsx file and insert the data into the web application. This thesis focuses on the development of the 3D image.

1.5 Initial approach

The development team had previous experience in web development. However, neither team member had previously worked with complex mathematical optimization or 3D visuals in web applications. The objective was to start with programming the base features, which was considered an easily achievable task, and then move onto studying and finally implementing the required features.

2 3D TECHNOLOGIES

Rendering 3D graphics in a web browser environment is not an entirely new technology. According to Desandro (n.d.), it has existed for years, in multiple varying forms. However, some technologies are less ideal for the problem the developer may be facing, and, thus, the technology used in development should be chosen with care.

This chapter will provide a brief description of each of the four different technologies that can be used for the creation of 3D scenes in web applications. A simple scene with a single rotating green cube with a white background was made, as seen in Figure 1, and the process is explained here briefly. Because the scenes made with each technology look the same, there will be no figure for each scene separately. Based on the information obtained during the development of the scenes, the strengths and weaknesses of each technology are inspected. Finally, comparisons are made, and the technology best suited to solve the problem in the container optimization tool is chosen.

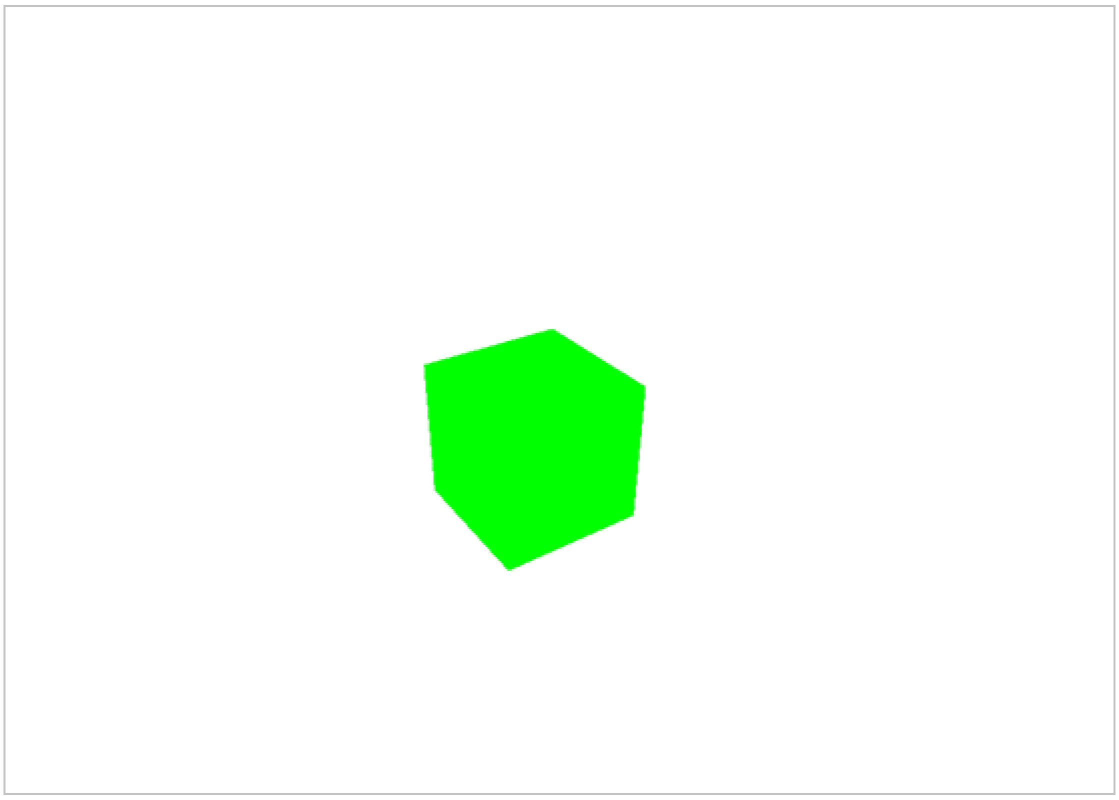


Figure 1. Three.js basic scene with a green cube.

2.1 CSS 3D transforms

CSS is generally used in stylizing HTML documents. However, even without any additional programming libraries or tools, CSS is capable of producing 3D graphics built into it. The CSS transform property provides a way to move and rotate a HTML element in three dimensions (Desandro n.d.).

CSS 3D Transforms have very little in terms of actual features for generating 3D scenes. The transform property's rotation and transformation functionalities are used to create, move and spin models. Textures are added by using the background image attribute. Any additional features such as camera controls or lighting must be created by the developer.

The creation of 3D scenes using this technology is achieved by using multiple div elements with the transform property to create each face of a 3D object. These faces will be assembled into the complete 3D model which can then be spun and moved inside 3D space using the translate and rotate attributes. (Clark, 2013). For example, as seen in Figure 2, a 3D model of a cube is created from six different faces.

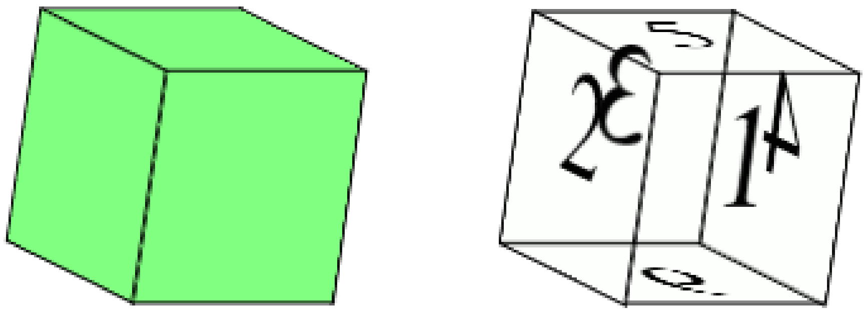
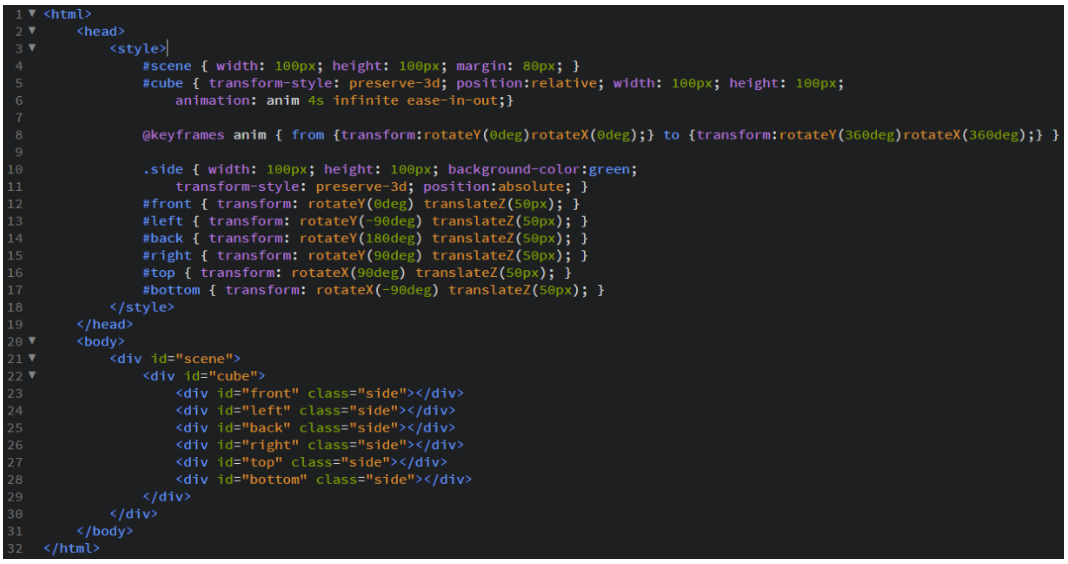


Figure 2. CSS 3D Transforms cube example.



```
<html>
    <head>
        <style>
            #scene { width: 100px; height: 100px; margin: 80px; }
            #cube { transform-style: preserve-3d; position:relative; width: 100px; height: 100px;
                animation: anim 4s infinite ease-in-out;}

            @keyframes anim { from {transform:rotateY(0deg)rotateX(0deg);} to {transform:rotateY(360deg)rotateX(360deg);} }

            .side { width: 100px; height: 100px; background-color:green;
                transform-style: preserve-3d; position:absolute; }
            #front { transform: rotateY(0deg) translateZ(50px); }
            #left { transform: rotateY(-90deg) translateZ(50px); }
            #back { transform: rotateY(180deg) translateZ(50px); }
            #right { transform: rotateY(90deg) translateZ(50px); }
            #top { transform: rotateX(90deg) translateZ(50px); }
            #bottom { transform: rotateX(-90deg) translateZ(50px); }
        </style>
    </head>
    <body>
        <div id="scene">
            <div id="cube">
                <div id="front" class="side"></div>
                <div id="left" class="side"></div>
                <div id="back" class="side"></div>
                <div id="right" class="side"></div>
                <div id="top" class="side"></div>
                <div id="bottom" class="side"></div>
            </div>
        </div>
    </body>
</html>
```

Figure 3. CSS 3D Transforms basic scene code example.

With sufficient prior experience in CSS, the development of a basic scene with a rotating cube is achieved easily. No programming is required as the elements need only be given specific attributes.

2.1.1 Advantages of CSS 3D Transforms

CSS 3D transforms are natively supported in nearly every modern web browser (CSS3 3D Transforms n.d.). This ensures the technology functions as desired without the need for additional extensions or plugins to be loaded. The technology functions entirely using CSS, so any programming can be kept completely separated from the 3D functionality if so desired. These benefits make this technology an excellent option for minor 3D effects used in enhancing web layouts, such as image carousels.

2.1.2 Disadvantages of CSS 3D Transforms

The creation of complex 3D objects is problematic as each face needs to be made as a separate div element. It is possible to move the model in the scene using CSS animation, but the lack of a skeleton significantly increases the workload required to animate a model as each face that requires stretching must be transformed separately. Attempts to move a single vertex causes each of the face's vertices to adjust, as the face's width or height must be changed.

CSS 3D transforms have very little in terms of features. The lack of even the most commonly available features such as cameras or lighting makes the technology strenuous if a larger task should be made with it, as these features must be implemented by the user.

2.2 WebGL 1.0

WebGL is an API used to render 3D graphics inside HTML documents. It allows web browsers to accelerate the rendering of the 3D graphics generated with it by using the device's hardware. Due to its flexibility, it can be used in the creation of any kind of 3D graphics. However, even the most basic features for 3D scenes such as creating objects or moving them must be programmed by the developer. WebGL is based on OpenGL ES 2.0 shader language, so it may be complex for developers only familiar with web development, and not with desktop 3D API. However, other libraries are built on top of this technology that simplify and accelerate development. (Jackson & Gilbert 2021. chapter 1: Introduction). Because the topic of WebGL is generally very broad, this thesis will only provide this brief description and further focus on two JavaScript libraries built on top of it.

2.2.1 Advantages of WebGL

At the time of writing this thesis, most modern browsers natively support WebGL 1.0 (WebGL - 3D Canvas graphics n.d.). WebGL is built into all browsers that support it, allowing it to be used without any other additional plugins for download or installation (Anyuru, 2012, 3). Because it behaves much like the API it is based upon, WebGL can be relatively easy to learn should the developer have any experience in using the OpenGL ES 2.0 (Anyuru, 2012, 3). Provided the device using WebGL has a powerful graphics processing unit, the API can render 3D graphics extremely quickly. This allows impressive 3D graphics to be displayed even in web environments.

2.2.2 Disadvantages of WebGL

The theory behind using WebGL can be overwhelming to an inexperienced user, and the requirement of writing the shaders in a completely different programming language may confuse the user further. The number of built-in functions that must be called in order to generate even the most basic scene make the API tedious to use and needlessly extend the time required in development.

The shaders that must be provided to the API must be written in GLSL shader language, so the knowledge of an additional programming language is required in order to fully utilize WebGL.

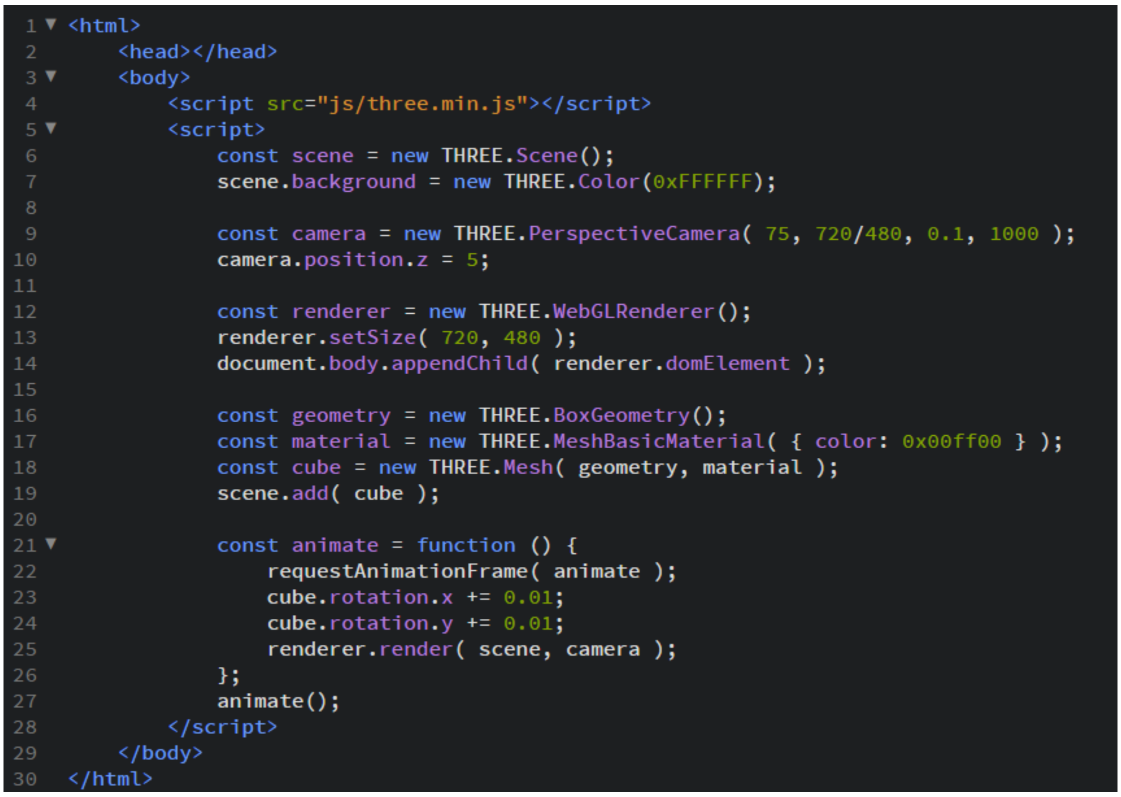
Rather than providing a method for transforming objects, the operation must be done through mathematical functions. This problem can be bypassed using an existing math library, but doing so forfeits the benefit of requiring no additional APIs or libraries. Other generally useful features for a 3D scene such as camera control or data loaders must be built by the developer or included using separate libraries.

2.3 JavaScript 3D Library Three.js

Three.js is a JavaScript Library used to render and animate 3D scenes in web browsers. It is based on WebGL and processes the graphics pipeline and mathematic functions required in building a 3D scene, simplifying the entire process for the user while also providing other features WebGL lacks by default, such as lighting and animations.

Because it is based on WebGL, Three.js provides the benefits of utilising the user's hardware to accelerate rendering. In addition, Three.js has features that can visually improve the 3D scenes such as post processing and lighting. Other features to speed up the development of 3D scenes are built into the API, such as animation, camera controls, data loaders, audio, materials, textures and various useful mathematical functions. Three.js also supports virtual reality by using WebXR.

Three.js must be run on a server due to browsers' same-origin policy security restrictions if any external files are loaded. This can be bypassed by using a local web server or changing the security files of the browser in use (How to run… n.d.). As Three.js is a JavaScript library, it must be loaded inside a script element. A renderer, camera and scene objects will be created, and the renderer will be placed inside a HTML element. Meshes are added into the scene after being created from a geometry and a material. The scene with all the changes made to it will be rendered again each time the render function is called. (Creating a scene n.d.) Three.js supports many different file types for 3D models, but the use of glTF is recommended because of its features and compact nature. These 3D models can be loaded into the scene using data loaders. (Loading 3D models three.js)



```html
<html>
    <head></head>
    <body>
        <script src="js/three.min.js"></script>
        <script>
            const scene = new THREE.Scene();
            scene.background = new THREE.Color(0xFFFFFF);

            const camera = new THREE.PerspectiveCamera( 75, 720/480, 0.1, 1000 );
            camera.position.z = 5;

            const renderer = new THREE.WebGLRenderer();
            renderer.setSize( 720, 480 );
            document.body.appendChild( renderer.domElement );

            const geometry = new THREE.BoxGeometry();
            const material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
            const cube = new THREE.Mesh( geometry, material );
            scene.add( cube );

            const animate = function () {
                requestAnimationFrame( animate );
                cube.rotation.x += 0.01;
                cube.rotation.y += 0.01;
                renderer.render( scene, camera );
            };
            animate();
        </script>
    </body>
</html>
```

Figure 4. Three.js basic scene code example.

The development of the basic scene in Three.js, for the purposes of this study, was quick. The low number of functions required in setting the scene up make the creation of a 3D scene a quick and easy process. The official documentation provides simple examples and excellent descriptions for the properties and functions of each class that help in most situations.

2.3.1 Advantages of Three.js

Three.js has access to WebGL's functionality of displaying 3D and accelerating rendering with hardware without the complexity the use of WebGL normally entails. If the user's browser does not support WebGL, the library has options for other renderers that can be used as a failsafe.

Three.js has a large number of built-in features that 3D applications can use, ranging from ways of improving the visuals of the 3D scene by means of PBR and post-processing effects, to ways of easing the programming process with pre-built data loaders and math utilities.

The excellent documentation features of Three.js makes it easy to learn to use the API. Each entry contains a description for what the object is used for, list of property and function descriptions, sample code and links to sample scenes which use the object.

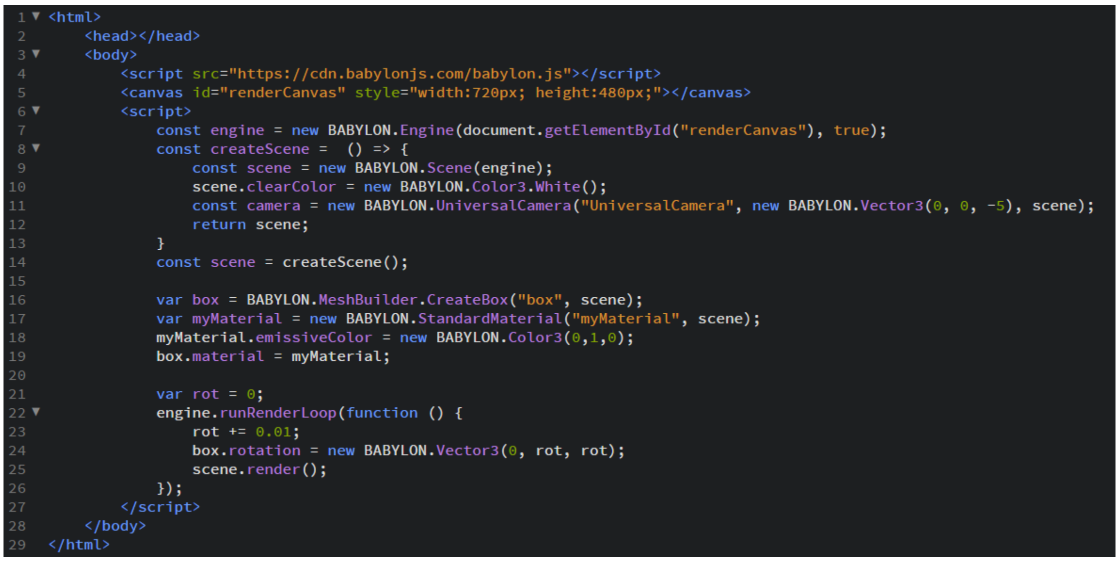
2.3.2 Disadvantages of Three.js

Three.js is frequently updated, and some syntaxes may change as new versions of the API are released. As a result, it may be detrimental to always rely on the latest version to ensure the application’s functionality in the long run unless maintenance is performed regularly. In addition, solutions for specific problems found by googling may not function properly with a given version.

2.4 JavaScript 3D Library Babylon.js

Babylon.js is a JavaScript game and rendering engine. It provides the user with rendering capabilities and includes physics, collisions and other features typically used in game development. It uses WebGL for rendering (Engine Specifications. n.d.).

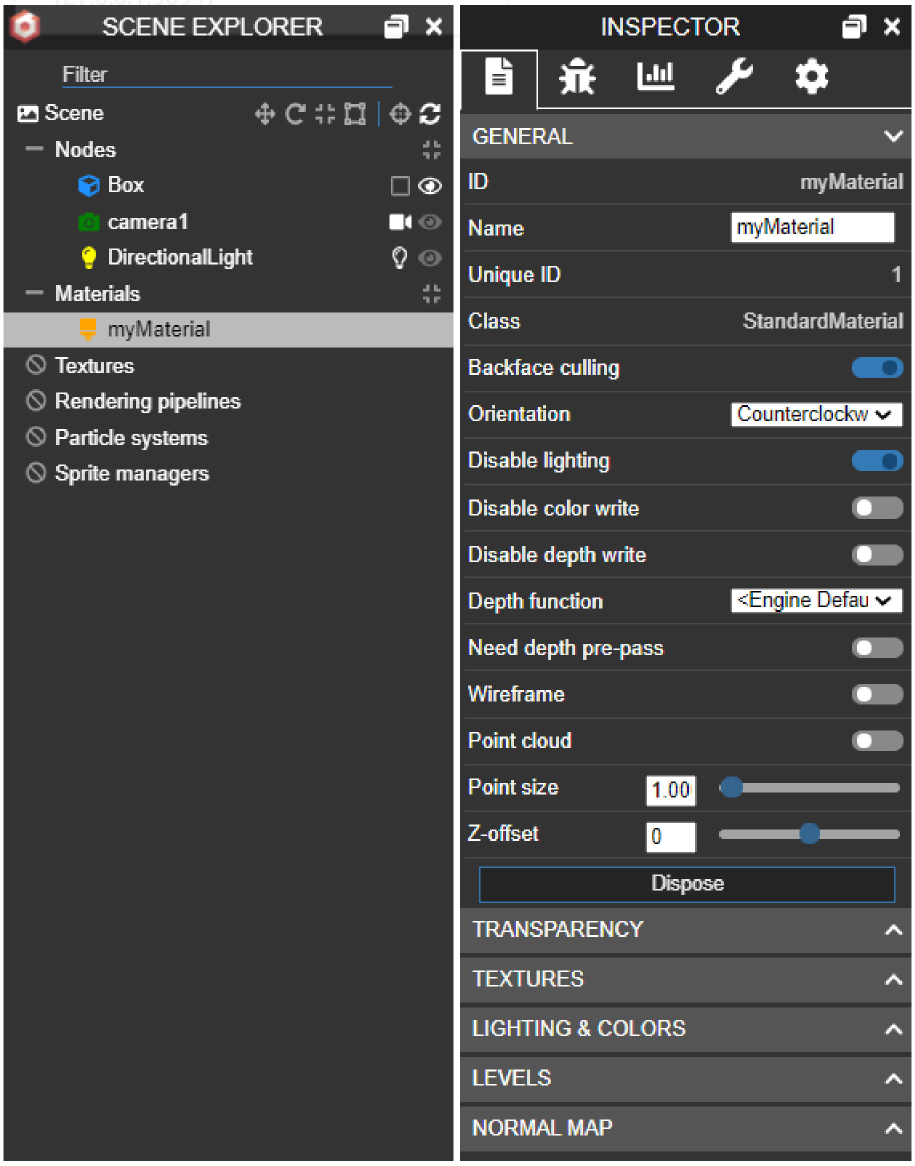
The process of setting up Babylon.js is somewhat similar to that of Three.js. The JavaScript library is loaded with a CDN link or locally after being downloaded from its GitHub page. A canvas element is created that will be used as the rendering viewport for the 3D scene, and the variables for the engine and scene are made. The engine is given the canvas element as a parameter, and then the scene is set to use the engine. Camera and any other objects can then be added into the scene. A render loop is made using the engine’s runRenderLoop function. (Babylon.js documentation n.d.)



```html
<html>
    <head></head>
    <body>
        <script src="https://cdn.babylonjs.com/babylon.js"></script>
        <canvas id="renderCanvas" style="width:720px; height:480px;"></canvas>
        <script>
            const engine = new BABYLON.Engine(document.getElementById("renderCanvas"), true);
            const createScene =  () => {
                const scene = new BABYLON.Scene(engine);
                scene.clearColor = new BABYLON.Color3.White();
                const camera = new BABYLON.UniversalCamera("UniversalCamera", new BABYLON.Vector3(0, 0, -5), scene);
                return scene;
            }
            const scene = createScene();

            var box = BABYLON.MeshBuilder.CreateBox("box", scene);
            var myMaterial = new BABYLON.StandardMaterial("myMaterial", scene);
            myMaterial.emissiveColor = new BABYLON.Color3(0,1,0);
            box.material = myMaterial;

            var rot = 0;
            engine.runRenderLoop(function () {
                rot += 0.01;
                box.rotation = new BABYLON.Vector3(0, rot, rot);
                scene.render();
            });
        </script>
    </body>
</html>
```

Figure 5. Babylon.js basic scene code example.

For the purposes of this study, the basic scene was quick to set up with the help of a sample code in the Babylon.js documentation. The functions are straightforward and easy to learn. The inspector feature was used to study the different available options in materials, but it was removed once the desired material was chosen. With the help of the inspector, any properties of any objects in the scene can easily be fine-tuned and any possible options can be further explored without the need to write new code, as seen in Figure 6.



ID	myMaterial
Name	myMaterial
Unique ID	1
Class	StandardMaterial
Backface culling	on
Orientation	Counterclockw
Disable lighting	on
Disable color write	off
Disable depth write	off
Depth function	<Engine Defau
Need depth pre-pass	off
Wireframe	off
Point cloud	off
Point size	1.00
Z-offset	0

Figure 6. Babylon.js Scene explorer and inspector.

2.4.1 Advantages of Babylon.js

Much like Three.js, Babylon.js is built on top of WebGL and enjoys most of its benefits. The similarities do not end there, as Babylon.js is very easy to use and learn owing to its instruction documentation and simplicity of methods. The library contains powerful tools for enhancing the look of a 3D scene, such as PBR, lighting, shaders, materials (Babylon.js n.d.). The inspector makes it easy to fine-tune and also serves as an excellent debugging tool. Babylon.js includes a physics engine as well, making it ideal for the creation of browser-based games.

2.4.2 Disadvantages of Babylon.js

Some of the sample code provided in the instruction documentation contain information that is irrelevant to the matter, which may make someone inexperienced with the software include code that is useless to the specific purpose they desire. The function for the creation of the scene is not included in the 'my first app' documentation either, which may cause further confusion. The library is larger than the other technologies explored in this thesis, but it is, nevertheless, a very small file to include.

2.5 Choosing a technology for the container optimization tool

The primary concerns in the development of the container optimization tool were its simplicity for the end user and the speed at which the software functions. The user interface was to be created outside the 3D scene, so the user interface's simplicity would not impact the choice of the 3D technology. This would leave the performance impact to be the client's only concern in the matter. However, development speed and ease of maintenance were also considered important aspects during the application's development.

Both CSS 3D transforms and WebGL provide the benefit of not requiring the loading of an additional API. However, development using these technologies is significantly slower in comparison to Three.js or Babylon.js. This is because 3D features such as camera movement and creation of shapes must also be programmed by the developer. In addition, the development effort already put into Three.js and Babylon.js makes them better optimized and more functional than entirely new code, which means that the rendering is faster as well. The loading of API occurs only when the application is started, whereas the rendering occurs multiple times a second whenever the camera is moved. Thus, the rendering has a greater impact on the application's performance, and the choice should be made between Three.js and Babylon.js.

Neither of the two technologies perform rendering noticeably quicker than the other, and in this thesis most of the visual effects and features exclusive to only one technology are considered redundant for the purposes of the web application in development. In this study, Three.js is chosen for the development of the container optimization tool due to its significantly lighter file size and ease of use.

3 IMPLEMENTATION

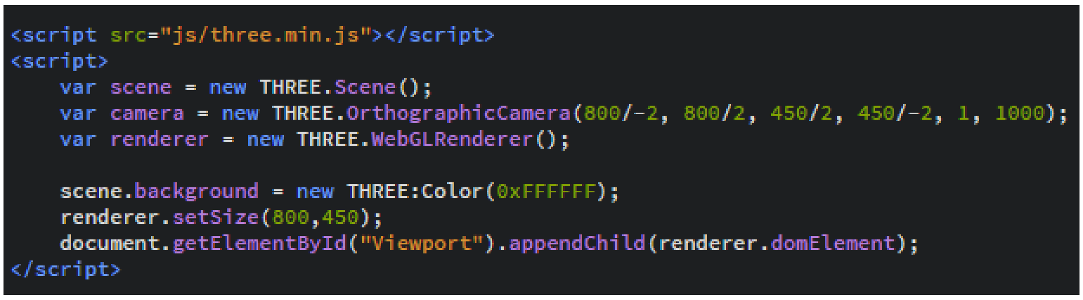
This chapter describes the implementation of the 3D scene inside the container simulator application as well as the choices made during the development. The implementation of the 3D scene began in June 2018 and ceased in 2019 July due to the completion of all features required at the time. The development resumed in May 2021 as the client requested the optimization functionality of pallets to be added. The creation of pallet functionality is not documented in this thesis.

Before the implementation of the 3D scene began, the application’s UI had been developed on the basis of the original mockup inside a HTML document. Fields where the user could input the dimensions of paper reels and the container had been given functionality using JavaScript, and any given data was stored as JavaScript objects. In addition, the optimization mathematic functionality had been developed to a point where the software could calculate the position for reels inside the container in a top-down perspective without factoring in stacking. The team had been using Brackets text editor in developing the project up to this point, and its live preview functionality creates a temporary local server. This would allow Three.js to function properly without requiring any further setup.

3.1 Beginning 3D development using Three.js

In order to begin developing the 3D scene, the latest revision of Three.js was downloaded from its GitHub page. The minimized source code was copied into the project folder, and the API was then loaded inside a script element. Another script element was made for all the programming related to Three.js to keep the file better organized. In addition, a new div element was made to contain the 3D scene.

Inside the new script element, variables for the scene, camera and renderer were made. The scene was given a white background color and a separate div element was made in the HTML file to contain the renderer. The renderer was then inserted into the div element using the appendChild method.



```
<script src="js/three.min.js"></script>
<script>
    var scene = new THREE.Scene();
    var camera = new THREE.OrthographicCamera(800/-2, 800/2, 450/2, 450/-2, 1, 1000);
    var renderer = new THREE.WebGLRenderer();

    scene.background = new THREE:Color(0xFFFFFF);
    renderer.setSize(800,450);
    document.getElementById("Viewport").appendChild(renderer.domElement);
</script>
```

Figure 7. Three.js renderer, scene and camera.

Three.js has a total of four different kinds of renderers, but the use of WebGLRenderer is highly preferred as the others lack support for some major features Three.js provides (CSS3DRenderer n.d.; SVGRenderer n.d.; CSS2DRenderer n.d.).

Three.js has five different kinds of cameras. Of these, the Perspective Camera and Orthographic Camera are the most commonly used and provide different ways of projecting the 3D scene into the rendering context. The remaining three camera types are multiples of perspective or orthographic cameras combined into a single object to better fit a specific situation, such as virtual reality. (ArrayCamera n.d.; CubeCamera n.d.; OrthographicCamera n.d.; PerspectiveCamera n.d.; StereoCamera n.d.)

In this study, the orthographic camera was chosen as the camera type used by the application as the clarity of measurements was considered more important than the more realistic look that a perspective camera with depth would provide.

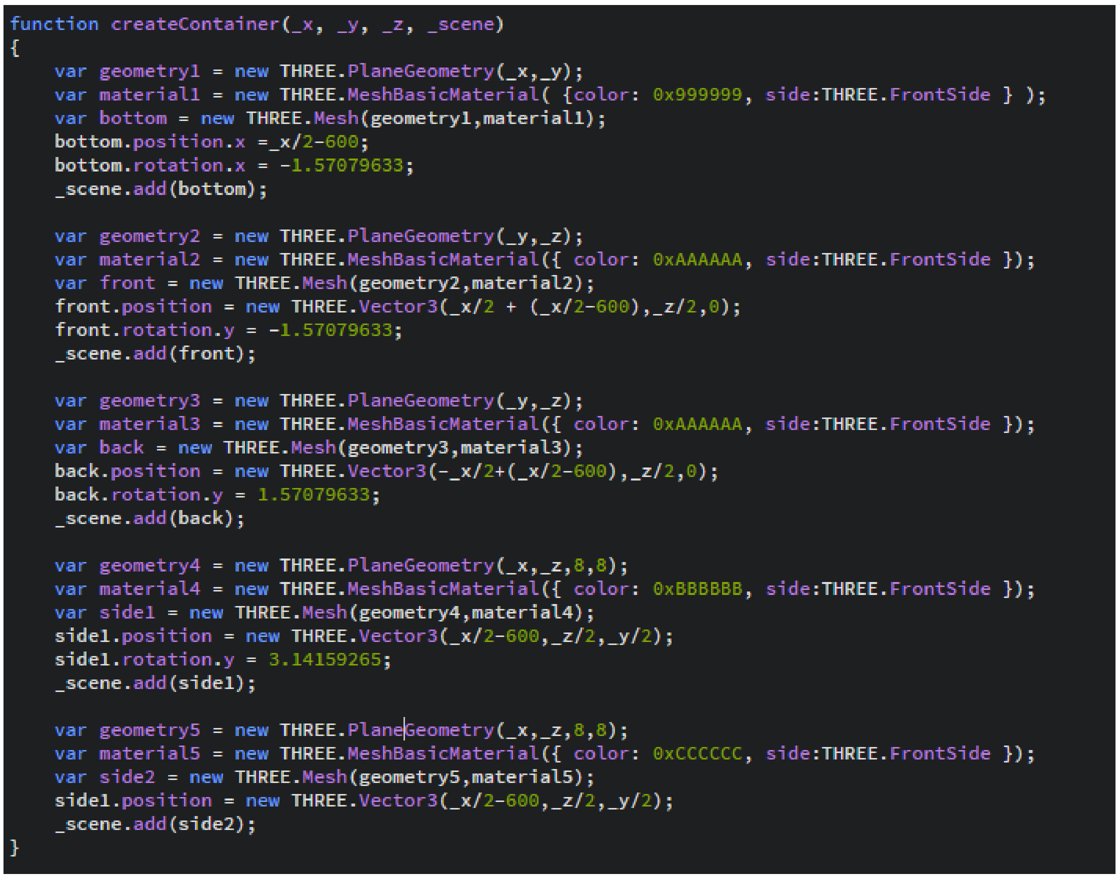
3.2 Creating the container

In order to display objects in the scene, they need to be created and added into it. Meshes are created from a single instance of a geometry and a material. Three.js provides numerous different kinds of geometries and materials.

Prior to the 125th release of Three.js, Three.js had a Geometry and a Buffer-Geometry class objects. The Geometry class offers lower quality in performance but is simpler to use (BufferGeometry n.d.). The Geometry class was deprecated by the developers of Three.js during the progress of writing this thesis, and the change was reflected in the 3D library's official documentation. Before updating to the latest version, most BufferGeometry generating syntaxes contained the word "Buffer". This was changed in the latest update. In order to match the new syntaxes, all lines in the code containing PlaneBuffer-Geometry were changed to PlaneGeometry. No further errors were caused by updating to the latest version. According to Herzog (2021), the Geometry class was removed entirely in the 125th release of Three.js to simplify various engine components and ease the maintenance of the engine's code.

Three.js has a total of 17 different materials that should cover every need the user might have, and in the case it does not, an entirely new material can be written using GLSL using the ShaderMaterial (ShaderMaterial n.d.). Mesh-StandardMaterial and MeshPhysicalMaterial are used with PBR, so these should be chosen if photorealistic graphics are required (MeshStandardMaterial n.d.; MeshPhysicalMaterial n.d.). Shadows and lighting are irrelevant for the Container optimization tool, so the best choice in materials is MeshBasic-Material which is not affected by lights (MeshBasicMaterial n.d.).

In the next phase of development, a function that creates the container using five one-sided PlaneGeometry geometries was made. Each of the geometries' see-through sides were set to be facing outwards to allow the reels inside the container to be seen from any direction, while also representing the container's walls and floor behind the reels. The materials used in the meshes were colored gray, and each plane was given a slightly varying tint of gray to provide clearer visuals. The dimensions for the container size were given as parameters, and correspond those used in real life.



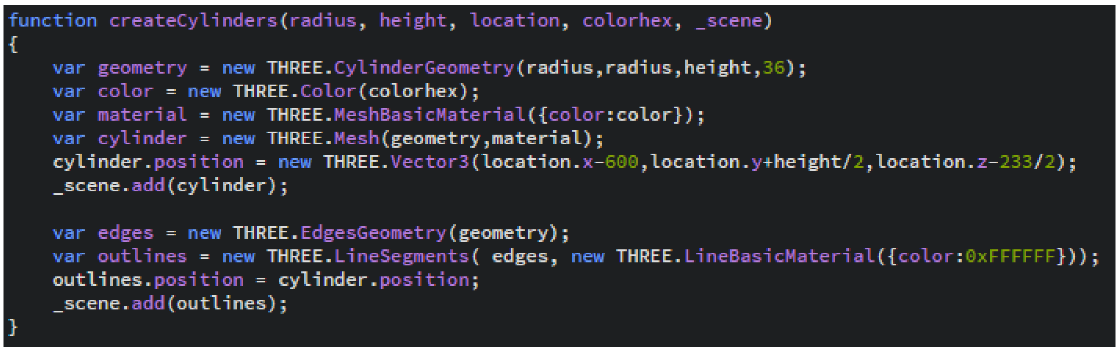
```
function createContainer(_x, _y, _z, _scene)
{
    var geometry1 = new THREE.PlaneGeometry(_x,_y);
    var material1 = new THREE.MeshBasicMaterial( {color: 0x999999, side:THREE.FrontSide } );
    var bottom = new THREE.Mesh(geometry1,material1);
    bottom.position.x =_x/2-600;
    bottom.rotation.x = -1.57079633;
    _scene.add(bottom);

    var geometry2 = new THREE.PlaneGeometry(_y,_z);
    var material2 = new THREE.MeshBasicMaterial({ color: 0xAAAAAA, side:THREE.FrontSide });
    var front = new THREE.Mesh(geometry2,material2);
    front.position = new THREE.Vector3(_x/2 + (_x/2-600),_z/2,0);
    front.rotation.y = -1.57079633;
    _scene.add(front);

    var geometry3 = new THREE.PlaneGeometry(_y,_z);
    var material3 = new THREE.MeshBasicMaterial({ color: 0xAAAAAA, side:THREE.FrontSide });
    var back = new THREE.Mesh(geometry3,material3);
    back.position = new THREE.Vector3(-_x/2+(_x/2-600),_z/2,0);
    back.rotation.y = 1.57079633;
    _scene.add(back);

    var geometry4 = new THREE.PlaneGeometry(_x,_z,8,8);
    var material4 = new THREE.MeshBasicMaterial({ color: 0xBBBBBB, side:THREE.FrontSide });
    var side1 = new THREE.Mesh(geometry4,material4);
    side1.position = new THREE.Vector3(_x/2-600,_z/2,_y/2);
    side1.rotation.y = 3.14159265;
    _scene.add(side1);

    var geometry5 = new THREE.PlaneGeometry(_x,_z,8,8);
    var material5 = new THREE.MeshBasicMaterial({ color: 0xCCCCCC, side:THREE.FrontSide });
    var side2 = new THREE.Mesh(geometry5,material5);
    side1.position = new THREE.Vector3(_x/2-600,_z/2,_y/2);
    _scene.add(side2);
}
```

Figure 8. createContainer function.

3.3 Creating the paper reels

In order to draw the paper reels into the 3D scene, a specific function was made. After receiving the dimensions, position and color as parameters, a CylinderGeometry was made with corresponding values, and a MeshBasicMaterial was made with the color parameter, and these two were used to form a complete mesh which was then added into the scene. In order to provide a better visualization of which paper reels belong to the same shipment, multiple reels were given the same color. The color was generated randomly using a separate function. With no lighting used in the scene, each reel was given flat colors. Because of the lack of lighting in the scene, when two reels of the same color are stacked on top of each other they may look like a single object rather than two separate ones. In order to ensure clarity, each reel was given clear white outlines using EdgesGeometry. The EdgesGeometry method uses a previously generated BufferGeometry class object’s shape to determine its location and other properties. In this case, the chosen BufferGeometry was the CylinderGeometry.

This createCylinder function is called by the optimization functionality. Once the calculations are finished, the cylinder drawing functionality is called with the dimensions, positions and colors of each reel inside the currently chosen container. The render function is performed after each reel is drawn.



```
function createCylinders(radius, height, location, colorhex, _scene)
{
    var geometry = new THREE.CylinderGeometry(radius,radius,height,36);
    var color = new THREE.Color(colorhex);
    var material = new THREE.MeshBasicMaterial({color:color});
    var cylinder = new THREE.Mesh(geometry,material);
    cylinder.position = new THREE.Vector3(location.x-600,location.y+height/2,location.z-233/2);
    _scene.add(cylinder);

    var edges = new THREE.EdgesGeometry(geometry);
    var outlines = new THREE.LineSegments( edges, new THREE.LineBasicMaterial({color:0xFFFFFF}));
    outlines.position = cylinder.position;
    _scene.add(outlines);
}
```

Figure 9. createCylinder function.

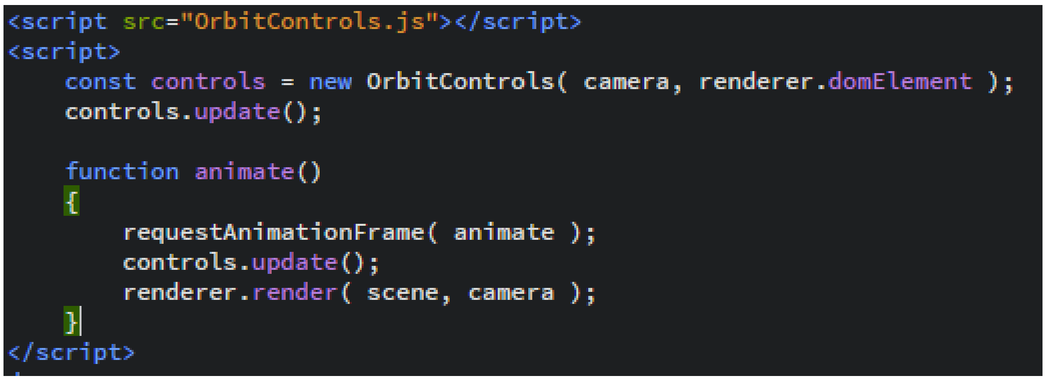
3.4 Camera controls

Three.js includes multiple different methods that allow the user to manipulate the camera. At the time of writing this thesis, eight different types of camera controls existed. They were all modular, and had to be either accessed with a CDN link, or downloaded from the GitHub page and then loaded separately into the HTML document with another script element.

Orbit Controls was chosen as the desired method of camera controls for the developed application, as the aim of the camera was to focus on and be rotated around the container. The Orbit Controls in the JavaScript portion was enabled by creating a new OrbitControls variable and calling its update function before calling the render function inside a render loop (OrbitControls n.d.).

3.5 Render loop

The render loop is used to render the scene again every time the screen is refreshed. The addition of camera control causes the scene to no longer be static, and the rendering must be done repeatedly. In order to achieve this, Three.js documentation recommends the use of requestAnimationFrame over other possible options (Creating a scene n.d.). The render function is called inside this to achieve the desired effect.

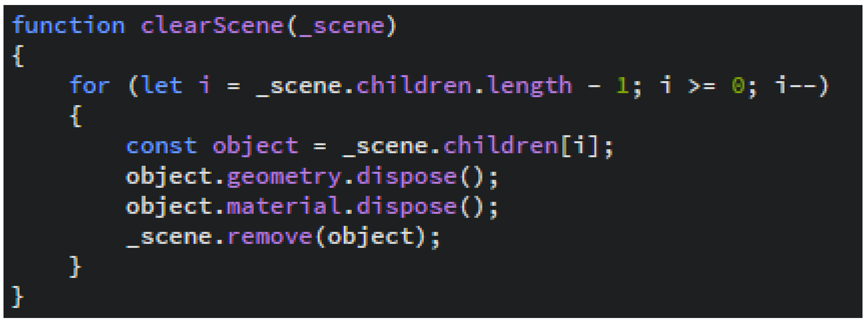


```
<script src="OrbitControls.js"></script>
<script>
    const controls = new OrbitControls( camera, renderer.domElement );
    controls.update();

    function animate()
    {
        requestAnimationFrame( animate );
        controls.update();
        renderer.render( scene, camera );
    }
</script>
```

Figure 10. Render loop and OrbitControls.

3.6 Clearing the scene

In most cases, all paper reels the user inputs into the application would not fit into a single container. The scene has to be cleared before displaying the next container because keeping hundreds of CylinderGeometry and LineGeometry objects in the memory could have more of an impact on the software's performance than generating twenty new ones. Three.js does not automatically release the memory occupied by unused geometries, materials, textures or render targets. This is because the scope of user-created entities, such as geometries and materials, is not known to the library. The automatic removal of objects not currently in use could lead to a disposal of objects that would be required during the next frame. This means objects must be released from the memory manually by means of the dispose method (How to dispose… n.d.). Each object created in the scene was cleared using a simple loop that would call the necessary dispose function. A new scene could then be rendered using the createContainer and createCylinder functions.



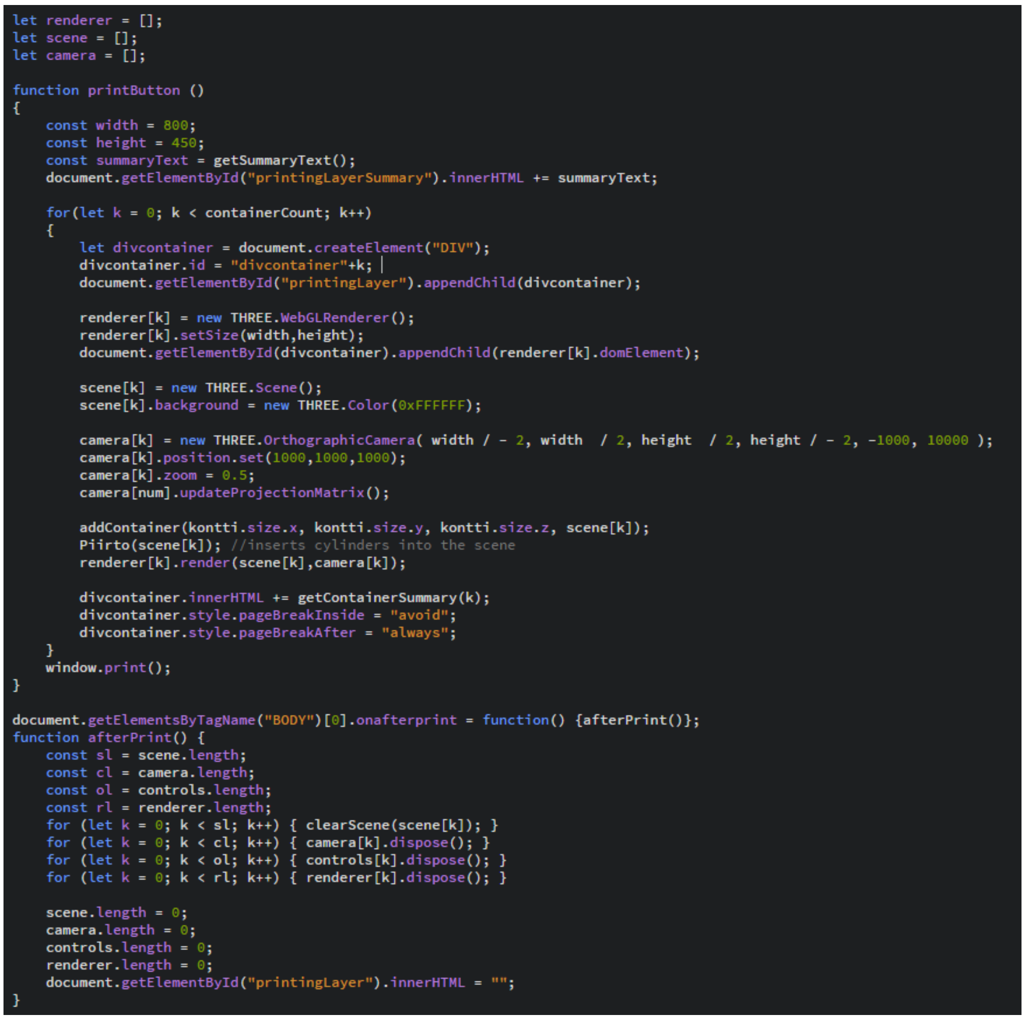
```
function clearScene(_scene)
{
    for (let i = _scene.children.length - 1; i >= 0; i--)
    {
        const object = _scene.children[i];
        object.geometry.dispose();
        object.material.dispose();
        _scene.remove(object);
    }
}
```

Figure 11. clearScene function.

3.7 Saving multiple renders into a single PDF file.

In the last phase of this thesis study, a function that would save all images of the containers with reels within them into a single pdf file was made.

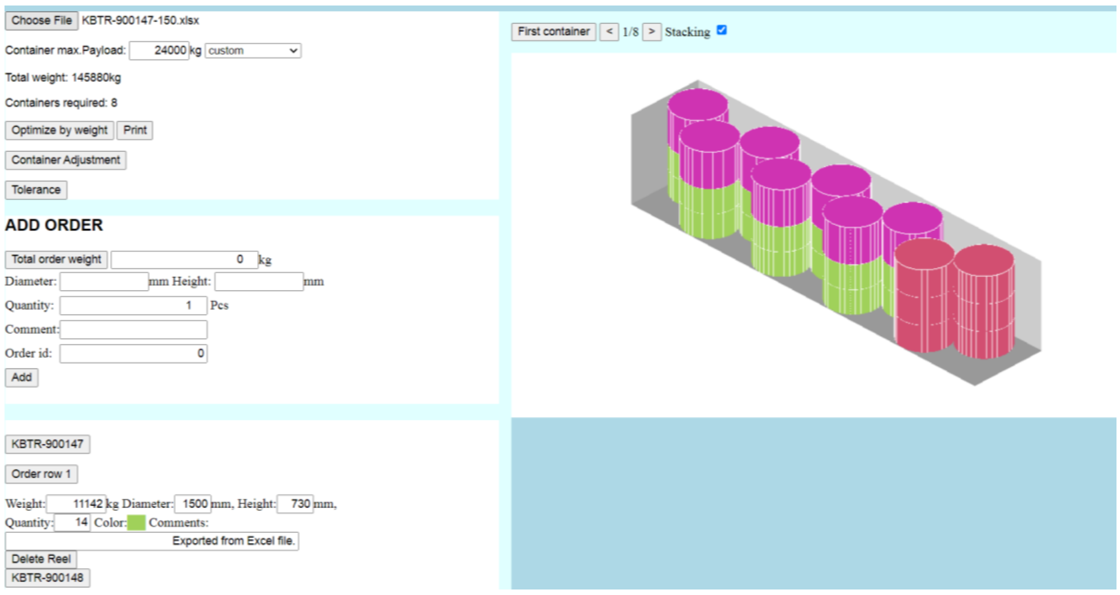
A completely new HTML layout was done for the printing, and the entire printing layout was hidden until printing was required. This was achieved easily using the CSS At-rules' @media print query, which can be used to apply CSS instructions only when the document is viewed in the print mode (MDN contributors, 2021.). In this case, the printable element was kept hidden using the display property. Because the number of required pages changes depending on the number of containers and their information, a JavaScript function was made that would create a canvas element to function as the render window. JavaScript was also used to create proper DOM elements to display information underneath the image. A single scene could not be used to render multiple different images, so every canvas element required its own scene. The window.print() function was used to first prompt the user with the printing window, from which it would be possible to save it as a .pdf file. Once the printing window was closed, the window.onafterprint method was used to clear the scene of any objects and DOM elements that would be only used only for printing.



```
let renderer = [];
let scene = [];
let camera = [];

function printButton ()
{
    const width = 800;
    const height = 450;
    const summaryText = getSummaryText();
    document.getElementById("printingLayerSummary").innerHTML += summaryText;

    for(let k = 0; k < containerCount; k++)
    {
        let divcontainer = document.createElement("DIV");
        divcontainer.id = "divcontainer"+k;
        document.getElementById("printingLayer").appendChild(divcontainer);

        renderer[k] = new THREE.WebGLRenderer();
        renderer[k].setSize(width,height);
        document.getElementById(divcontainer).appendChild(renderer[k].domElement);

        scene[k] = new THREE.Scene();
        scene[k].background = new THREE.Color(0xFFFFFF);

        camera[k] = new THREE.OrthographicCamera( width / - 2, width  / 2, height  / 2, height / - 2, -1000, 10000 );
        camera[k].position.set(1000,1000,1000);
        camera[k].zoom = 0.5;
        camera[num].updateProjectionMatrix();

        addContainer(kontti.size.x, kontti.size.y, kontti.size.z, scene[k]);
        Piirto(scene[k]); //inserts cylinders into the scene
        renderer[k].render(scene[k],camera[k]);

        divcontainer.innerHTML += getContainerSummary(k);
        divcontainer.style.pageBreakInside = "avoid";
        divcontainer.style.pageBreakAfter = "always";
    }
    window.print();
}

document.getElementsByTagName("BODY")[0].onafterprint = function() {afterPrint()};
function afterPrint() {
    const sl = scene.length;
    const cl = camera.length;
    const ol = controls.length;
    const rl = renderer.length;
    for (let k = 0; k < sl; k++) { clearScene(scene[k]); }
    for (let k = 0; k < cl; k++) { camera[k].dispose(); }
    for (let k = 0; k < ol; k++) { controls[k].dispose(); }
    for (let k = 0; k < rl; k++) { renderer[k].dispose(); }

    scene.length = 0;
    camera.length = 0;
    controls.length = 0;
    renderer.length = 0;
    document.getElementById("printingLayer").innerHTML = "";
}
```

Figure 12. printImages and window.onafterprint functions.

After the printing method was built, the development of the 3D scene was finished within the framework of this thesis study. Figure 13 displays the state of the application in June 2019 when the development of the 3D scene ended.



KBTR-900147-150.xlsx

Container max.Payload: 24000 kg custom

Total weight: 145880kg

Containers required: 8

ADD ORDER

Total order weight 0 kg
Diameter: mm Height: mm
Quantity: 1 Pcs
Comment:
Order id: 0

Weight: 11142 kg Diameter: 1500 mm, Height: 730 mm,
Quantity: 14 Color: 🟩 Comments:
Exported from Excel file.

1/8

Figure 13. Finished 3D scene.

4 CONCLUSION

The purpose of this thesis was to study different ways of implementing 3D graphics in a web application and choose the most suitable one for the needs of a particular software. From the available alternatives, Three.js was chosen to be the most fitting 3D technology. The required 3D scene was made, and its functionality was tied to the mathematical calculations done by the application.

Due to excellent documentation, the development of the 3D scene using Three.js began swiftly and progressed at a quick pace all the way to the project's end. Whenever a problem was encountered, a quick look at the corresponding documentation page helped to solve the issue, and the development could continue.

The documentation of the development process can be used as a general guide on generating a 3D scene using Three.js. Certain aspects of the generated scenes were examined in general terms to introduce other available options. In addition, a general overview with the advantages and disadvantages of three other technologies was provided.

The study of different 3D graphics technologies provided a great amount of insight into how they function both with and independent of web applications. Web 3D technologies are so powerful their capabilities rival desktop applications, and with certain libraries 3D scenes can be generated in web applications easily.

The use of 3D graphics opens many doors in game development. However, displaying 3D graphics has applications beyond just entertainment purposes. Not only can they make a web page more engaging and interesting, they also improve the visual image of the application compared to using only 2D graphics. The experience gained during this thesis study will be an asset in future projects.

REFERENCES

Anyuru, A. 2012. Professional WebGL Programming : Developing 3D Graphics for the Web. John Wiley & Sons, Incorporated.

Babylon.js. n.d. Engine Specifications. WWW-Document. Available at: https://www.babylonjs.com/specifications/ [accessed 5 May 2021].

Babylon.js documentation. n.d. Getting Started - Chapter 1 - Setup Your First Web App. WWW-Document. Available at: https://doc.baby-lonjs.com/start/chap1/first_app [accessed 5 May 2021].

Can I use. n.d. CSS3 3D Transforms. WWW-Document. Available at: https://caniuse.com/#feat=transforms3d [accessed 11 September 2020].

Can I use. n.d. WebGL – 3D Canvas graphics. WWW-Document. Available at: https://caniuse.com/webgl [accessed 11 September 2020].

Clark, K. 2013. Creating 3D worlds with HTML and CSS. WWW-Document. Available at: https://keithclark.co.uk/articles/creating-3d-worlds-with-html-and-css/ [accessed 11 September 2020].

Desandro, D. n.d. Intro to CSS 3D transforms. WWW-Document. Available at: https://3dtransforms.desandro.com/ [accessed 11 September 2020].

Herzog, M. 2021. THREE.Geometry will be removed from core with r125. WWW-Document. Available at: https://discourse.threejs.org/t/three-geometry-will-be-removed-from-core-with-r125/22401 [accessed 7 March 2021].

Jackson, D. & Gilbert, J. 2021. WebGL Specification. Khronos Group. WWW-Document. Available at: https://www.khronos.org/registry/webgl/specs/lat-est/1.0/ [accessed 8 May 2021].

Kotkamills, n.d. Company. WWW-Document. Available at: https://kotkamills.com/company/ [accessed 11 September 2020].

MDN contributors. 2021. @media. WWW-Document. Available at: https://de-veloper.mozilla.org/en-US/docs/Web/CSS/@media [accessed 11 May 2021].

Three.js documentation. n.d. ArrayCamera. WWW-Document. Available at: https://threejs.org/docs/?q=camera#api/en/cameras/ArrayCamera [accessed 11 September 2020].

Three.js documentation. n.d. BufferGeometry WWW-Document. Available at: https://threejs.org/docs/?q=bufferge#api/en/core/BufferGeometry [accessed 11 September 2020].

Three.js documentation. n.d. Creating a scene. WWW-Document. Available at: https://threejs.org/docs/?q=creating#manual/en/introduction/Creating-a-scene [accessed 11 September 2020].

Three.js documentation. n.d. CSS2DRenderer. WWW-Document. Available at: https://threejs.org/docs/?q=renderer#examples/en/renderers/CSS2DRenderer [accessed 11 September 2020].

Three.js documentation. n.d. CSS3DRenderer. WWW-Document. Available at: https://threejs.org/docs/?q=renderer#examples/en/renderers/CSS3DRenderer [accessed 11 September 2020].

Three.js documentation. n.d. CubeCamera. WWW-Document. Available at: https://threejs.org/docs/?q=camera#api/en/cameras/ArrayCamera [accessed 11 September 2020].

Three.js documentation. n.d. How to dispose of objects. WWW-Document. Available at: https://threejs.org/docs/#manual/en/introduction/How-to-dispose-of-objects [accessed 11 September 2020].

Three.js documentation. n.d. How to run things locally. WWW-Document. Available at: https://threejs.org/docs/#manual/en/introduction/How-to-run-things-locally [accessed 11 September 2020].

Three.js documentation. n.d. Loading 3D models. WWW-Document. Available at: https://threejs.org/docs/#manual/en/introduction/Loading-3D-models [accessed 11 September 2020].

Three.js documentation. n.d. MeshBasicMaterial. WWW-Document. Available at: https://threejs.org/docs/?q=material#api/en/materials/MeshBasicMaterial [accessed 11 September 2020].

Three.js documentation. n.d. MeshPhysicalMaterial. WWW-Document. Available at: https://threejs.org/docs/?q=material#api/en/materials/MeshPhysicalMaterial [accessed 11 September 2020].

Three.js documentation. n.d. MeshStandardMaterial. WWW-Document. Available at: https://threejs.org/docs/?q=material#api/en/materials/MeshStandardMaterial [accessed 11 September 2020].

Three.js documentation. n.d. OrbitControls. WWW-Document. Available at: https://threejs.org/docs/?q=orbitcont#examples/en/controls/OrbitControls [accessed 11 September 2020].

Three.js documentation. n.d. OrthographicCamera. WWW-Document. Available at: https://threejs.org/docs/?q=camera#api/en/cameras/OrthographicCamera [accessed 11 September 2020].

Three.js documentation. n.d. PerspectiveCamera. WWW-Document. Available at: https://threejs.org/docs/?q=camera#api/en/cameras/PerspectiveCamera [accessed 11 September 2020].

Three.js documentation. n.d. StereoCamera. WWW-Document. Available at: https://threejs.org/docs/?q=camera#api/en/cameras/StereoCamera [accessed 11 September 2020].

Three.js documentation. n.d. SVGRenderer. WWW-Document. Available at: https://threejs.org/docs/?q=renderer#examples/en/renderers/SVGRenderer [accessed 11 September 2020].

Figures