SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

**Aleksandr Smelov**

# Integration of interactive 3D models into React-based application

Bachelor Thesis
Autumn 2024
Bachelor of Engineering, Automation Engineering

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

## Thesis abstract

Degree Programme: Bachelor of Engineering, Automation Engineering

Specialisation: Machine Automation

Author: Aleksandr Smelov

Title of thesis: Integration of interactive 3D models into React-based application

Supervisor: Kauppinen, Raine

Year: 2024 Number of pages:47 Number of appendices:0

This thesis explores the possibilities of integrating interactive 3D interfaces into React-based web applications, utilizing specialized tools and techniques to enhance web applications through the integration of 3D elements, while the primary objective was to explore the capabilities of React as a foundation for developing 3D web applications, with a focus on creating visually engaging and responsive 3D content.

The thesis examined several prominent 3D libraries and tools, such as Three.js, React Three Fiber (R3F), Spline, Babylon.js and p5.js, analysing their unique features, limitations and compatibility with the React ecosystem, and conducting a comprehensive comparison, thus providing a basis for evaluating each tool's effectiveness, functionality and user interaction. Ultimately, the project aims to identify the most efficient and scalable solutions for building interactive 3D experiences in React environments.

This thesis highlights the potential of interactive 3D interfaces to enhance web applications through a comprehensive research and development process. The finalized application, designed for desktop use and hosted on GitHub Pages, illustrates various implementations of 3D elements within a React framework. This work adds to the knowledge of 3D development tools and methods specifically concerning React, serving as a valuable resource for both developers and researchers eager to explore advancements in interactive 3D web technologies.

[1] Keywords: React, 3D Basics, Shaders, Primitives, Web-interface

# TABLE OF CONTENTS

## Contents

## Terms and Abbreviations

| | |
|---|---|
| 2D | Form that may be described in two dimensions, Two-dimensional. |
| 3D | Form that may be described in three dimensions, Three-dimensional. |
| API | Application Programming Interface, a set of rules or protocols that enables software applications to communicate with each other. |
| BEM | Block-Element-Modifier CSS methodology used for structured application of styles to markup. |
| CSS | Cascading Style Sheets, programming language describing styles applied to the markup of application. |
| DOM | Document Object Model, the data representation of the objects that comprise the structure and content of a document on the web. |
| GitHub | A developer platform that allows developers to create, store, manage and share their code. |
| GLSL | OpenGL Shading Language, a high-level shading language with syntax based on the C programming language. |
| HTML | Hypertext Markup Language, used for building interface markup. |
| IDE | Integrated Development Environment, a software application that provides comprehensive facilities for software development. |
| JS | JavaScript, a programming language allowing to implement complex features on web pages. |
| Node.js | A free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts. |

| | |
|---|---|
| npm | Node Package Manager, a default package manager for the JavaScript runtime environment Node.js |
| OpenGL | Open Graphics Library, a cross-language, cross-platform API for rendering 2D and 3D vector graphics. |
| React | Declarative component-based JavaScript library allowing developers to build reusable UI components and optimizing rendering performance by following the Virtual DOM approach. |
| R3F | React Three Fiber, a React renderer for Three.js. |
| UI | User Interface, the part of application visible to users. |
| URL | Uniform Resource Locator, the address of a unique resource on the internet. |
| UX | User Experience, a cumulative of relevance, interactivity and performance of application, describing user's satisfaction. |
| VDOM | Virtual Document Object Model, is a programming concept where a virtual representation of a UI is kept in memory and synchronized with the real DOM by a library such as React. |
| Vite | A frontend tool for quick building of web applications |
| VS Code | A streamlined code editor with support for development operations like debugging, task running, and version control |
| WebGL | Web Graphics Library, a JavaScript API for rendering high-performance interactive 3D and 2D graphics within any compatible web browser without the use of plug-ins. |
| WebXR | An API for web content and apps, allowing to use the interface with mixed reality hardware such as VR headsets and glasses with integrated augmented reality features. |

## Figures

# 1. INTRODUCTION

## 1.1 Background

The digital landscape has changed due to the quick development of online technologies, which have increased hardware efficiency and raised demand for 3D models that provide better user engagement and comprehension. This is especially noticeable in industries like e-commerce and education, where 3D visuals provide a higher level of immersion than conventional interfaces. By allowing users to engage with 3D models, people can practice processing complex data virtually and intuitively. In fields like engineering and design, where better decision-making is facilitated by real-time visualizations, this skill is vital. Furthermore, 3D models in education improve understanding of difficult ideas while accommodating a variety of learning preferences and encouraging student participation.

The simultaneous development of WebGL and related technologies has made it possible to deliver extremely immersive 3D material through web browsers, eliminating the need for additional plugins. Both the growing need for creative web design and developments in the area have enabled this democratization of 3D material. Consequently, this trend has fostered the development of various extensible libraries that facilitate the integration of 3D models into websites.

Despite the availability of advanced features, challenges remain in optimizing performance and ensuring compatibility across various browsers that will still be used. This underscores the intricate challenges designers and developers face in delivering an exceptional user experience (UX), often with minimal interaction elements. Understanding the best practices and techniques for incorporating 3D models into these frameworks is becoming more and more important as interactive web apps continue to evolve.

This thesis aims to contextualize the current state of integrating interactive 3D models into modern web-based applications. Moreover, it will explore the technological landscape alongside design principles that encourage effective implementation.

## 1.2 Research Goals and Objectives

The primary goal of the thesis is to develop a comprehensive web application that effectively utilizes the features of a 3D interface, specifically designed for desktop resolutions. To achieve this, a detailed comparative analysis of various development tools, utilities, and libraries will be conducted. This analysis will evaluate their effectiveness, capabilities, and relevant use cases, ultimately informing best practices for selecting and using resources related to 3D application development.

Another key objective is to ensure an optimal user experience through a well-designed user interface. This interface will promote intuitive and predictable navigation while ensuring fast performance in response to user interactions with the 3D elements of the application.

## 1.3 Structure

The thesis provides a detailed account of the basic principles of graphic-powered three-dimensional space, involving elementary units of which 3D models are constructed, information about shaders, and related terms. Furthermore, consideration is given to the tools empowering for interactive 3D development with an accompanying comparison of their efficiency and compatibility with React. The development process described further justifies tools used for creating specific sections of the website, including figures representing the output.

# 2. OVERVIEW OF THREE-DIMENSIONAL SPACE

## 2.1 Coordinate systems

In three-dimensional space coordinate systems are used as a structure according to which the locations of points and objects are specified. Currently, there are three such systems which have grown popular in the development of 3D interfaces.

In the Cartesian coordinate system that many people rely upon today three dimensions are described by employing three perpendicular axes – x, y, and z – for each dimension correspondingly (Eldridge, n.d). The location of a point in this system is expressed by the triplet (x,y,z), where each coordinate corresponds to the distance along one of three axes from the point of their intersection or "zero-point". Such a straightforward method proves to be useful for depicting space in fields such, as computer graphics and physics for its efficient spatial representation.

Another significant coordinate system is the **spherical** coordinate system, which originates from the polar coordinate system by extending it into three dimensions. In this coordinate system a point's position in space is defined by three variables: the distance $\rho$ along the radial line connecting the point to the fixed point of origin or model's center point, the polar angle $\theta$, the angle between the radial line and a given polar axis, and the azimuthal angle $\varphi$, the angle of rotation of radial line around a given polar axis (Encyclopedia Britannica, 2024a). This coordinate system proves to be advantageous in representing points on orbits or spherical surfaces.

The Cylindrical coordinate system is also commonly used in three-dimensional modelling. It combines aspects of both Cartesian and polar coordinates, as in this system a point's position in space is defined by distance ρ along the radial line coming from the origin or model's center point, the angular coordinate φ, and the height z of the point over the base axis (Weisstein, n.d.-a). Cylindrical coordinates are widely used in engineering applications, where objects often obey cylindrical symmetry, such as pipes or columns.

It is crucial for efficient development and performance to choose the appropriate coordinate system while representing or modelling 3D interfaces, as the Cartesian coordinate system may be simple for the transformation or transition of models linearly, while the spherical

system enhances the development of circular or orbital transitions with precise rotational symmetry.

Moreover, many applications for graphical editing, such as Blender or Spline, integrate multiple coordinate systems. This combination may facilitate interactions between objects, simplify the positioning of models in complex scenes and use the benefits of all coordinate systems integrated for precise rotational and linear translations. This adaptability is vital for the development of applications providing immersive UX where users can intuitively interact with the 3D environment.

## 2.2 3D model basics

A 3D model is constructed from various fundamental components defining its structure and appearance in three-dimensional space. Visual representation of an object depends on these components, therefore affecting further rendering, manipulability and interactivity of the model. Understanding these elementary principles is crucial for working with 3D models.

### 2.2.1 Vertices and Edges

A vertex is the simplest element of a 3D model representing a point in 3D space. According to the Cartesian coordinate system, it is commonly defined by its coordinates along the X, Y, and Z axes. Vertices do not exist in isolation but are meant to be connected forming edges and faces (Weisstein, n.d.-b). The number of vertices in a model largely determines its complexity and shape, as the higher number of vertices causes a higher number of edges and faces to be produced, which, therefore, can provide detailed and smooth surfaces but may also lead to low performance while processing a model.

Edges, on the other hand, connect two vertices, forming a straight line between them. Multiple edges define the structure of the model, creating a mesh necessary for forming a visible surface geometry, while edges themselves do not appear visible to a user. A 3D model containing only vertices and edges would behave as a wireframe or skeleton, lacking any surface detail or texture.

### 2.2.2 Faces (polygons)

Faces or polygons are crucial for building the 3D model, as they form a visible surface within the spaces between three (triangular polygon) or four (quadrilateral polygon) edges (Encyclopedia Britannica, 2024b). The surface of a model having more polygons (hi-poly) will appear smoother and more precise in comparison to the same model built with a low number of polygons (low-poly). However, an excess of faces increases the complexity of the model and raises computational requirements for its processing, so models are often optimized by reducing the number of faces if possible.

### 2.2.3 Normals

Normals are vectors associated with faces or vertices that indicate the direction in which the surface is oriented and are necessary for determining the interactions of light with the surface of the chosen polygon (Thorne, 2024). By averaging the normals of neighbouring faces smooth shading between them may be achieved, thus enhancing UX by creating a realistic appearance where the light will reflect differently depending on the orientation of faces.

### 2.2.4 UV Coordinates

UV coordinates are required for the mapping of 2D textures onto a 3D model’s surface. The model’s surface, if unfolded, may be represented as a 2D net, where each point on the net corresponds to a specific point on a 2D texture image, thus allowing the texture to wrap around a model. The letters U and V refer to the axes in the 2D texture space analogously to the X, Y, and Z axes in 3D space (Wikipedia, n.d). It is important to define UV coordinates properly, as otherwise the image may be applied with distortion affecting the UX, while correctly defined UV can result in achieving clean and realistic texturing.

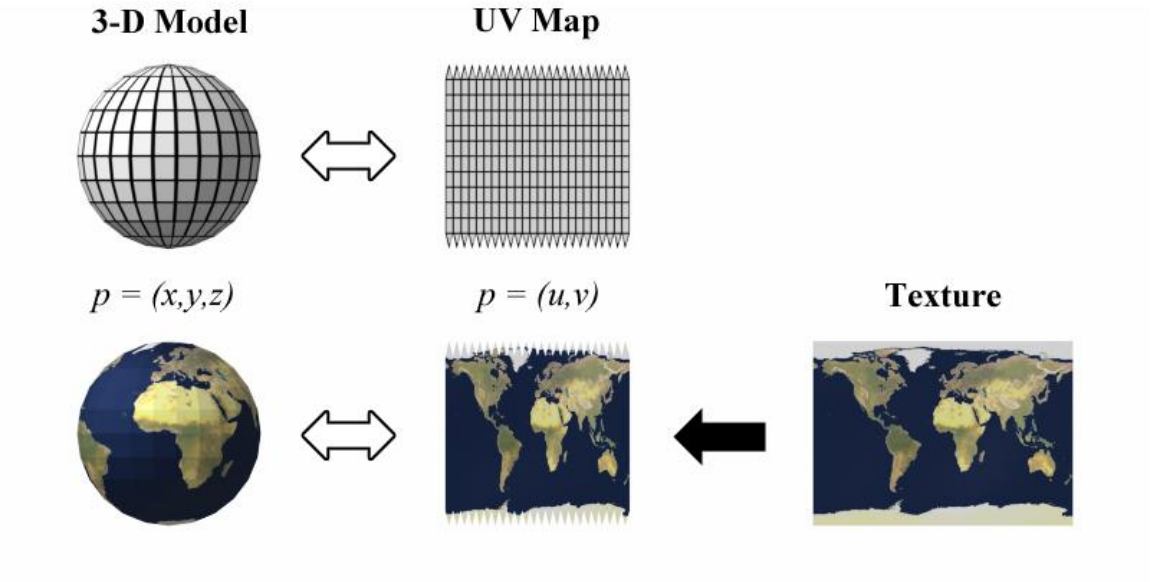


Figure 1. The application of a texture in the UV space related to the effect in 3D (Wikipedia, n.d)

### 2.2.5 Center Point (Origin)

The center point, also known as the origin, is a reference point according to which the location, rotation, and scaling of a model in 3D space are defined. The center point can be local (specific for each model) or global (the origin of the whole scene). The local center point may be shifted from the actual center of the object or even beyond the object, thus rotating the model will produce an orbiting effect instead of smooth rotation around a fixed point in the center of an object.

### 2.2.6 Bounding Box

The bounding box is an invisible box enclosing the entire 3D model or a group of objects. It is used to define the spatial extent of the model, making it easier to determine the position of an object or detect possible intersections with other objects in a scene. The bounding box is particularly useful for approximate calculations of the model's size, as modern 3D editors provide features allowing one to see the edges of a bounding box while manipulating the object or group of objects.

When combined, the aforementioned elements form the foundation of any 3D model. By carefully manipulating them, developers can create clean and visually compelling objects. Understanding these elements is essential for the successful integration of 3D models into

the interface of any application, as it provides the elementary basics for controlling how the models are rendered, interacted with, and optimized within a dynamic, interactive environment.

### 2.3 Rendering

Rendering in computer graphics is the process of generating 2D images from 3D models by simulating how light interacts with objects in a scene (Verma & Walia 2010, pp.29-33). This process involves complex calculations that determine the colors, shadows, reflections, and textures, transforming raw 3D data into a 2D representation on the screen. Rendering can be classified into two categories: real-time and pre-rendering. Real-time rendering aims to produce images rapidly for dynamic, interactive experiences while pre-rendering focuses on achieving high levels of realism, often used in film production and detailed visualizations.

Rendering can be broadly divided into two categories: real-time rendering and pre-rendering. Real-time rendering aims to produce images instantly, with high frame rates, allowing for dynamic, interactive experiences. This is essential for applications like games or interactive web interfaces, where the scene needs to be updated constantly as users interact with it. Pre-rendering rendering, on the other hand, is slower but achieves high levels of realism and is often used in film production and detailed visualizations. Balance of quality and performance is a key aspect of rendering, as it affects the visual appeal and responsiveness of the final application.

### 2.4 Shaders

Shaders are specialized programs running on the GPU and controlling how 3D models are rendered in real-time applications defining how light, color, and texture interact with each element of the model (OpenGL Wiki, 2019). Shaders are usually broken into two types commonly used in the rendering pipeline – vertex shaders and fragment shaders. In web development, both shaders are written in GLSL and work jointly to produce the final visual output on the screen.

### 2.4.1 Vertex shaders

Vertex shaders affect specific vertices of a 3D model by shifting their position from the initial state of the mesh according to the algorithm described in the .glsl format (OpenGL Wiki, 2017). Besides positioning, vertex shaders also rearrange additional data, like normals (for lighting) and UV coordinates (for texturing), to sequentially pass it to the fragment shader for further processing. Vertex shaders are essential for animation effects, adding realism to a model's movement or deformation.

### 2.4.2 Fragment shaders

Fragment shaders (or pixel shaders) run right after the vertex shaders, calculating the color of each pixel that will be rendered on the model's surface (OpenGL Wiki, 2020). They define the interactions of a surface with light based on the details of texturing, lighting, and shading and determine whether a surface looks rough, shiny, transparent, or textured based on the data passed from the vertex shader. For instance, fragment shaders can compute how light scatters or reflects off the surface, creating effects like diffuse lighting or more complex mirror effects, requiring higher processing power.

Moreover, shaders have a customisation feature enabling developers to create specific visual effects. For instance, a custom vertex shader might be used to alter vertices dynamically resulting in smooth animation, while a fragment shader could apply complex materials giving a model such features as transparency or reflectivity. Such tools as WebGL and Three.js make it easier to implement custom shaders by providing a framework to manage low-level operations, therefore developers can write custom vertex and fragment shaders to bring interactivity and realism into their 3D interfaces

The interaction between vertex and fragment shaders is crucial for rendering, as while the vertex shader defines the geometry and structure of a model, the fragment shader determines the final appearance of its surface. Together, they form a complete pipeline converting a 3D model into a fully rendered object on the screen

## 2.5 Light

In building 3D interfaces lighting plays a vital role in creating depth and realism, giving a visual accent to a model and providing the overall atmosphere in a scene. Light is the major aspect defining how the shape, texture, and material applied to a model are perceived by the user, as when light hits an object it acts depending on the properties applied to the surface, refracting, reflecting or scattering (Dimitrijević et al., 2013, p. 115). The elementary principles of light in 3D rendering often refer to simulating real-world behaviours such as intensity, direction, color, falloff angle or ability to cast shadows, which highlight models giving them volume and form.

Light is typically processed through a shader program that calculates the effect of the light falling on a specific point or fragment, defining its color, brightness or even visibility. By choosing different types of light and adjusting their properties, developers can precisely control the illumination of a 3D scene. The most popular types of light used in 3D development are described further.

### 2.5.1 Ambient Light

Ambient light illuminates all objects in the scene equally with given intensity and color from every side, simulating the diffuse light emitted from every point in the scene (Dimitrijević et al., 2013, p. 116) Ambient light is functionally the simplest type of lighting, as it has no direction and does not cast shadows, which makes it useful as an initial level of light preventing objects from appearing too dark. But, unlike other lights, it lacks the realism of making an accent on an object's shape and can make a scene appear flat if it is the only light used in the scene.

### 2.5.2 Directional Light

Directional light simulates parallel rays of light coming from a distant source, which, however, cannot be positioned to any definite coordinates within the scene, but can be emitted from the chosen direction (Dimitrijević et al., 2013, pp. 115-116). Directional light’s principle is to illuminate all objects from the same angle, regardless of their position in space, mimicking how sunlight affects large outdoor scenes and casting shadows essential for creating depth and dimension highlighting the shapes of objects.

### 2.5.3 Point Light

Point light emits light in every direction from a single point positioned within the scene and, generally, has a similar behaviour to a regular light bulb (Dimitrijević et al., 2013, p. 115). The key feature of this light is a diminishment of its intensity with distance in accordance with the inverse-square law (Figure 2), which states that the intensity is inversely proportional to the square of the distance from the source of light, therefore this effect makes objects close to the point light appear brighter, while distant objects get darker. Point lights cast soft shadows similar to real-world light sources, such as lamps or candles adding depth and realism to a scene.

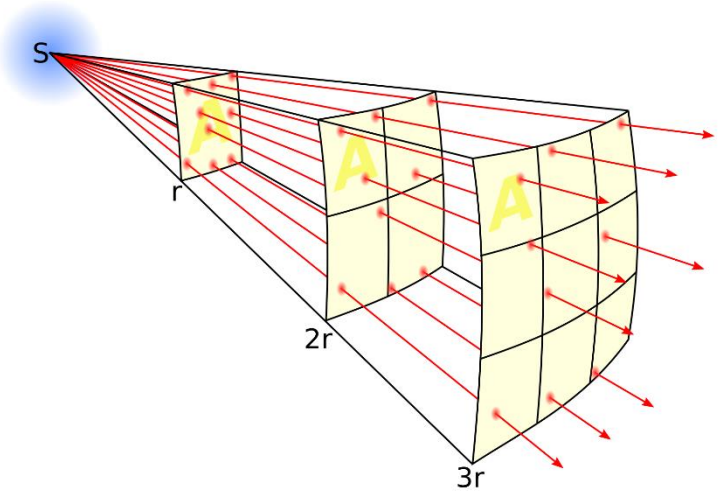


Figure 2. Inverse-square law (Wikipedia, n.d)

### 2.5.4 Spot Light

Spot lights emit light within a cone-shaped area with such parameters as position, direction, and angle further defining the illuminated region (Dimitrijević et al., 2013, p. 115). Spotlights produce concentrated beams of light also characteristic of flashlights or car headlights, illuminating specific areas within a cone while leaving the surrounding space in darkness. The spotlight's intensity can be adjusted to fade with distance, as well as the feature of casting shadows can be implemented also, allowing for lighting effects highlighting specific objects.

### 2.5.5 Hemisphere Light

Hemisphere lighting replicates smooth outdoor illumination by providing soft, diffused light both from above and below an object. By utilizing two colors, the first of which represents light from the sky and the other from ground reflections, such an implementation creates a natural gradient enhancing the realism in outdoor scenes (Three.js, n.d.-a). Although it does not cast shadows similar to ambient light it seamlessly blends light across objects, making it a choice for background and environmental lighting.

Concluding the above mentioned, light in 3D rendering refers to simple principles: the direction, intensity, and ability to cast shadows or interact with materials define how the user perceives objects. Whether simulating natural sunlight with directional lights or creating localized illumination with point or spotlights, each type of light serves a specific function. By understanding and utilizing these principles, developers can create dynamic 3D environments that engage and immerse the viewer.

## 2.6 Camera

In 3D rendering, the camera acts as the observer's viewpoint, akin to a traditional camera. It determines the visible sections of the 3D scene by enclosing objects within the frame, the parameters of which dictate how the user perceives the scene. The camera plays a crucial role in defining the perspective, depth, and shape of the window containing the rendered scene and the overall composition of the render.

There are two main types of cameras used in 3D applications:

1. Perspective Camera: This camera emulates human visual perception of the world, whereby proximate objects appear larger and distant objects smaller, thus enhancing a sense of depth (Three.js, n.d.-b). Perspective cameras are frequently employed in scenes mimicking realistic environments to maintain natural spatial relationships among objects.
2. Orthographic Camera: In contrast to a perspective camera, an orthographic camera renders objects without perspective distortion, resulting in the uniform size of objects, regardless of distance (Three.js, n.d.-c). This type of camera is commonly employed in technical or architectural visualizations requiring precise measurements not affected by the influence of perspective.

Cameras in 3D applications can undergo such transformations as movements, rotations, or zooming, thereby providing flexible visualization. By manipulating the camera's properties developers can shape UX and highlight specific aspects of the scene. Additionally, settings such as clipping planes play a critical role in determining which parts of the scene are rendered, promoting efficient performance by excluding objects beyond the visible proximity thresholds.

# 3. TECHNOLOGIES OVERVIEW

## 3.1 React

### 3.1.1 Introduction to React

React, the well-known JavaScript framework was first launched in 2013 by Facebook (currently Meta) for building user interfaces, respectively for online applications (Gackenheimer, 2015). React's efficiency, modular design, and ability to update certain user interface elements instead of reloading the entire page are features that make it appealing to a broad programming community, while the Component-based architecture turns React into an indispensable tool for developers striving towards more organized and manageable code and user interface broken into smaller, reusable parts called components (Chen et al., 2019, pp. 119-120).

However, there are other popular frameworks which are continuously supported and can be an alternative to React:

- Angular.js developed by Google Inc. is an opinionated, complete MVC framework providing developers with tools to build and style interfaces, bind data, inject dependencies, set up routing, handle forms, and more. Angular follows a two-way data binding approach, where changes in the UI automatically update the DOM and vice versa. On the other hand, Angular.js is not very performance-efficient in comparison to React, especially in large-scale applications (Capała & Skublewska-Paszkowska 2018, pp. 82–86)
- Vue.js is an open-source framework which, similarly to React, is oriented on work with the view-layer of an application (view-layer-oriented), it has the same component-based architecture and uses a two-way data binding approach as Angular.js (Bielak et al., 2022, p. 79). However, Vue.js, as an open-source project, faces challenges related to funding, which hinders it from sustainable improvement and modernization, therefore, it is more likely to be used in smaller projects than React.

React is a highly popular and powerful tool and is widely used in prominent technology companies like Netflix, Airbnb, Uber and Meta for their front-end development due to its

flexibility and scalability. Currently, React is the key component of many contemporary development stacks, it has expanded beyond just a UI toolkit thanks to its actively expanding ecosystem of tools, libraries, and community support.

### 3.1.2 Virtual DOM and Component-Based Structure

One of React's notable innovations is the Virtual DOM (VDOM). The regular DOM is a model similar to a folder tree in a computer's file explorer generated by the browser according to the received by a web page HTML code which is used to render its content. Regular updates to the DOM, like adding or removing elements are slow and resource-intensive in traditional web applications. React avoids this problem by using a virtual representation of the DOM, which functions as an in-memory "shadow" copy of the actual DOM (Aggarwal, 2018, p.133). React updates the VDOM when a component's state changes rather than directly modifying the browser's DOM, and then compares the VDOM and the real DOM (a process known as reconciliation) to find differences.

During reconciliation, React calculates the minimal number of updates to be committed and executes those adjustments in the actual DOM. React lowers the overhead use of DOM manipulation by committing only the necessary changes, leading to more optimised performance of applications which therefore get quicker and more responsive (Chen et al., 2019, p. 125). This method works especially well for apps with intricate user interfaces because the user experience would be significantly slowed down if the full page were updated after each change.

React’s component-based structure enhances the efficiency of the application even further. A component in React can be thought of as an independent module that contains a section of the user interface (React, n.d.-a). Each component has its own functions, state(s), and contained user interface, and can be reused multiple times throughout the application, even in other components. Such a modular approach greatly simplifies the development process, allowing developers to build UIs out of reusable, manageable parts having their own responsibilities. As a result, this method reduces the amount of written code.

When a component’s properties (props) or state(s), on which component’s DOM content depends, change, React automatically triggers a re-render of only that component or parts of it, rather than reloading the entire page (React, n.d.-b). This selective re-rendering, combined

with the VDOM, provides the high performance of even complex applications. Additionally, React's component-based architecture which encourages the separation of code into components, makes the application more convenient for maintenance and development.

### 3.1.3 States and Hooks

Management of the state of a component is another of React's key features for building interactive applications. State represents the dynamic data of a component, that might be changed under specific circumstances defined by a developer (React, n.d.-c). Upon any modification of the state, React re-renders the component to reflect the updated state, ensuring that the UI, dependent on state variables, is synchronised with the underlying data.

Initially, management of state variables was available only in React's class components by the employment of the *setState* method. The advent of Hooks in React 16.8 marked a significant advancement in the framework, allowing functional components to utilize state variables. Thus, revolutionizing development practices driving a more intuitive approach to building user interfaces. Hooks empower developers to incorporate state and various React functionalities without addressing class components (React, n.d.-d), resulting in cleaner, more succinct code. This shift towards functional components streamlines the overall development process, enhancing productivity and efficiency in React applications.

The most common hook for managing state is *useState*. It allows functional components to maintain the local state. *useState* hook takes an initial state value as an argument and returns an array containing the current state variable and a function employed to update it (React, n.d.-e).

```
function Counter(){
  const [count, setCount] = useState(0);

  return(
      <div>
          <button onClick={() => setCount(count + 1)}>Click Me</button>
          <p>The button was clicked {count} times</p>
      </div>
  )
}
```

Figure 3. use of useState within a functional component

In the example shown in the Figure 3, *count* is the current state, and *setCount* is its update function. Whenever *setCount* is called, React re-renders the component to reflect the new value of count.

Another frequently used hook is *useEffect*, which enables developers to run a function contained in it as a callback after the render is complete, ensuring that the UI is updated before any side effects occur (React, n.d.-f). Side effects include actions like fetching data from an API, updating the DOM manually, running other functions or changing states. The *useEffect* hook also accepts a dependency array of state variables controlling when the callback should be re-run. If any of the variables in the array change, the callback is triggered again.

```
function Counter(){
  const [count, setCount] = useState(0);

  useEffect(() => {

    console.log(count);

  }, [count]);

  return(
    <div>
      ...
    </div>
  )

}
```

Figure 4. use of useEffect within a functional component

The example in Figure 4 shows that each time the state variable *count* is changed *useEffect* hook triggers and runs its callback function, thus the content of count is being printed in the console.

React provides several other widely used hooks, including:

- *useContext*: Allows components to access the global state from a context provider, making it easier to manage global data without passing it through several components as props. (React, n.d.-g)

- *useReducer*: An alternative to *useState* for managing more complex state logic. Here, instead of replacing the current state, the incoming value is first modified by the `reducer` argument – a function which processes new value and replaces the current state with the result of processing. (React, n.d.-h)
- *useMemo* and *useCallback*: The working principle of these hooks is similar to *useEffect*, as they are invoked after component rendering and dependency changing. However, unlike *useEffect*, *useMemo* and *useCallback* return values – variable and function respectively. Upon initialisation, these hooks execute a function passed as the argument and store the result in the value on which the hook was utilised. On subsequent re-renders of the component they return the last computed value unless a dependency changes, which triggers evaluation. (React, n.d.-i, n.d.-j)

The utilization of hooks facilitates the management of a component state and side effects in a clear and declarative manner, enhancing the efficiency and maintainability of React applications. Furthermore, hooks ease the transition from class-based to functional components, streamlining modern React development.

## 3.2 3D Implementation Tools

### 3.2.1 Three.js

Currently, Three.js is one of the most popular choices for the development and integration of 3D interfaces into web applications. It offers a high-level API for creating animated 3D graphics using WebGL technology (Evans et al., 2014, pp. 7-9). This enables extensive customisation which is ideal for complex 3D applications that are also interactive but despite that fact, it is complex to set up and requires deep knowledge of its principles such as model-lifetime, and syntax. Three.js is not specific to React but on the other hand, is widely supported and can be combined with React using tools like R3F (React Three Fiber).

### 3.2.2 React Three Fiber

React Three Fiber being an enhanced version of Three.js is designed specifically for the integration of 3D within React applications and encourages React's declarative syntax and

management of 3D elements as components, providing intuitiveness of use and efficiency in the integration of complex 3D scenes (React Three Fiber.docs, n.d). R3F is particularly powerful because it directly leverages Three.js, bringing the full feature set into the React ecosystem, delivering high performance even in complicated applications and handling 3D components with improved ease of use and reusability.

### 3.2.3 P5.js

P5.js, as Sandberg (2019, p. 42) concluded, is an open-source coding library focused on creating visual effects, drawing, 2D transitions, and although it’s not primarily a 3D engine, it includes some WebGL features such as animations and the use of shaders which work well for simpler projects. P5.js’s integrity in relation to React is considerable, as it relies on its own state management and render loop, which consequently may cause conflicts with React’s re-render practices. Nevertheless, p5.js is effective for lightweight interactive 3D animations and provides easy-to-learn syntax making it a choice for beginners.

### 3.2.4 Babylon.js

Babylon.js is a high-performance 3D rendering engine suitable for both WebGL and WebXR applications. It includes a comprehensive suite of tools for handling physics, animation and advanced rendering effects typical for game development and high-resolution scenic transitions (Babylon.js, n.d).

Babylon.js is provided with online and desktop scene editors incorporating the node geometry editing system, flexible positioning and styling tools which are characteristic for complex modelling utilities, and code export functionality empowering developers to embed the scene into a JavaScript project. Nevertheless, this framework isn't React-oriented as well as P5.js, therefore it requires several code adjustments and installation of additional npm libraries. Despite Babylon.js being a powerful tool for editing and integration of complex, professional-looking scenes with extensive documentation and community support, it requires expertise in working with its 3D modelling tools and code syntax, which makes it not a ‘beginner-friendly’ tool. Moreover, Babylon.js is less effective, than its alternatives, in terms of performance. For instance, Babylon.js consumes 46% more memory in comparison with Three.js according to Johansson (2021, p. 32)

### 3.2.5 Spline

Spline is the newest among the aforementioned tools. It was launched in 2021 and received broad popularity in 2023. Spline is appealing to designers and developers as it enables them to create 3D scenes with minimal coding through an intuitive, visual interface available as a desktop and an online application offering plenty of useful features like real-time team collaboration, 3D Modelling, simple assignment of responses of models on interaction, layer system of materials and more (Spline, n.d). Spline prioritizes ease of use, allowing users to design, develop and export interactive 3D scenes directly on multiple platforms (web, android, iOS, web-constructors) or as an embeddable code for different frameworks. While it doesn't integrate deeply with React like R3F, Spline scenes can be integrated into React applications as iframes or components, making it a great option for teams focused on visual design without extensive coding requirements (op. cit.).

### 3.1.7 Summary

Summarising the topic of 3D development tools, it is important to notice that the comparison of tools relies mostly on their compatibility with React, which is chosen as a development tool for the current thesis's practical part described in Chapter 4.

For React-specific 3D development, React Three Fiber (R3F) stands out due to its seamless integration with React's ecosystem, which, on the other hand, is not typical for complicated tools such as Three.js and Babylon.js despite their underlying power for complex customizations. P5.js better suits creative 2D projects, although it has some WebGL-related functionality. But regarding React it is the least compatible tool because of its own state management system and reload-loop. Spline, while being not directly React-oriented, excels in ease of design-focused 3D implementation with minimal coding and provides developers with exclusive creative features and the tool for multi-platform export, making Spline a preferable choice for development along with R3F.

## 3.3 Deployment Technologies

In the context of web development, the term "deployment" is used to describe the process of making a web application accessible via a server and a designated domain. Deployment

tools facilitate efficient publishing and automate setup tasks, thereby enabling developers to focus their attention on feature development.

### 3.3.1 GitHub Pages

GitHub Pages is a widely utilised, cost-free service for the hosting of static websites, which can be accessed directly from a GitHub repository. The deployment of an application to GitHub Pages necessitates the installation of the “gh-pages” npm package and the implementation of specific modifications to the package.json file. Once the configuration has been completed, developers are enabled to deploy their applications directly from the terminal of their IDE. This approach is particularly well-suited to smaller projects and personal portfolios.

### 3.3.2 Netlify

On the other hand, Netlify offers a more dynamic deployment solution, providing the ability to perform continuous deployment from a directly linked GitHub repository (Netlify, n.d). By connecting a GitHub account, developers can configure Netlify to automatically build and deploy their applications, whenever a new code was pushed to the repository. Furthermore, this platform permits the use of custom URL links, which is advantageous for projects in the development or testing stage.

Both GitHub Pages and Netlify facilitate the deployment process, but they are designed for different purposes. GitHub Pages service is well-suited to static websites with limited backend functionality, whereas Netlify supports continuous integration and deployment, making it more suitable for projects that require regular updates. The choice between the two depends on the complexity of the project and the frequency of updates.

# 4. DEVELOPMENT

## 4.1 Initialisation

### 4.1.1 Kickstart

To kickstart the development process, the project was initialized with React and Vite – a fast build tool that manages the application's quick setup and offers a hot module replacement feature that applies code changes in the running application for an efficient development experience. Before starting, Node Package Manager (NPM) was installed, as it’s essential for managing dependencies and embedding third-party libraries and extensions into projects. For code management the Visual Studio Code (VS Code) IDE was employed, this setup provided an ideal foundation for a smooth development process.

To set up a React project using Vite, the initialization command was run in the terminal (Figure 5).

```
# Initialize a new React project with Vite
npm create vite@latest my-3d-app -- --template react
```

Figure 5. Terminal command for initialization of React/Vite application directory

This command initializes a React application template configured with Vite in the directory currently opened for modification in VS Code. This setup allows the project to use React as the core framework by providing node modules necessary for React’s performance while leveraging Vite’s rapid build times. By following the commands described in Figure 6

the working directory was changed to the recently initialized folder, required node dependencies were installed, and the development server was started.

```
# Navigate to the project directory
cd my-3d-app

# Install dependencies
npm install

# Start the development server
npm run dev
```

Figure 6. Terminal commands launching the React/Vite application locally

Once the server was running, the application could be viewed in the browser at the localhost URL displayed in the Terminal until the runtime of the command “npm run dev” is interrupted manually or the VS Code IDE is closed.

### 4.1.2 Styling

For structuring CSS styles the BEM (Block-Element-Modifier) methodology was employed, as it ensures that styles are organized, scalable and reusable. According to BEM, the interface must be broken into blocks, elements, and modifiers, thus this approach promotes a consistent and readable CSS structure. Moreover, the BEM methodology restricts developers from using the id attributes of HTML tags for styling, thereby keeping the CSS written according to the class attributes only.

In BEM, a *Block* is an independent component (like a “menu” or “button”), an *Element* represents a part of a block with no standalone meaning (like “menu__item” within a “menu” block), and a *Modifier* is a variation of either a Block or Element (such as menu__item_selected where `selected` is Modifier). This hierarchy results in CSS classes that are descriptive and avoid conflicts.

```html
<div class="menu">
  <div class="menu__item menu__item_selected">Home</div>
  <div class="menu__item">About</div>
</div>
```

Figure 7. HTML code styled according to BEM methodology

```css
.menu {
  /* Styles for the menu block */
}

.menu__item {
  /* Styles for each item within the menu */
}

.menu__item_selected {
  /* Styles for the selected item */
}
```

Figure 8. CSS classes named according to BEM methodology

The BEM, while used in every CSS file in the application, keeps styles modular and predictable, thus providing ease of maintenance and scaling of the project as it grows. This methodology is especially useful in complex applications where reusable and clear CSS structures are essential.

## 4.2 Web application

The application was organized into distinct sections, each concentrating on specific aspects of 3D modelling. This approach facilitated a structured and interactive experience for users while offering valuable insights into the workflow.

### 4.2.1 Introductory section

The introductory section of the application was designed using Spline, emphasizing an interactive and visually engaging 3D experience through animations responsive to user

interactions (Figures 9, 10). The design enhances user engagement by integrating animations that respond to specific behaviors, such as hovering and clicking.

In this section, each 3D model has multiple states that transition seamlessly in response to user interactions. For instance, distinct animations or visual transformations are activated by clicking or hovering over different models, thus enhancing interactivity and providing immediate feedback, enriching the user experience (Figure 11). Furthermore, smooth transitions guide user attention and promote exploration within the 3D environment.

In addition to animations, Spline empowers developers to embed hyperlinks in models, which trigger upon clicking the model and redirect the user to external sources.

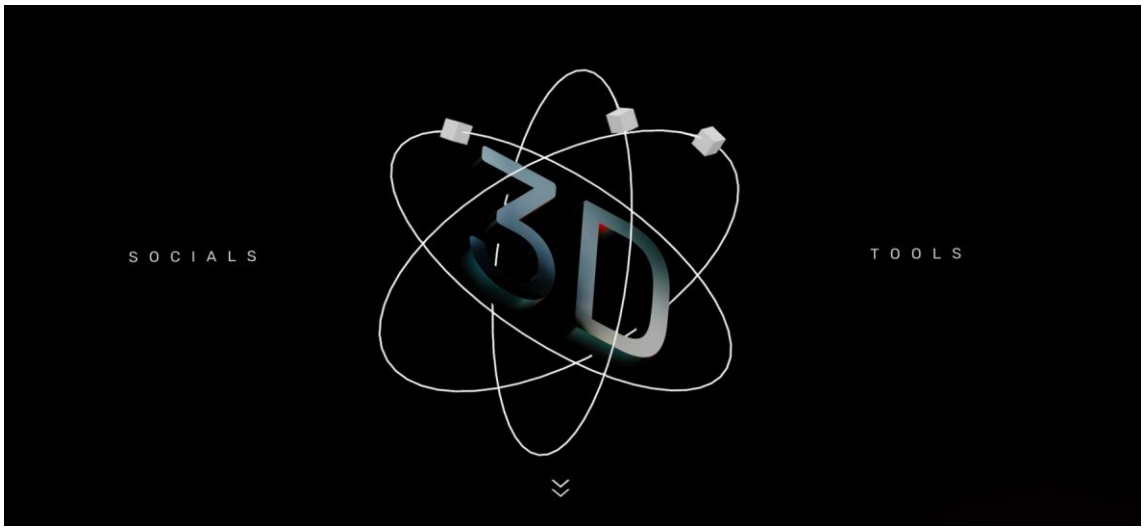


Figure 9. Intro section

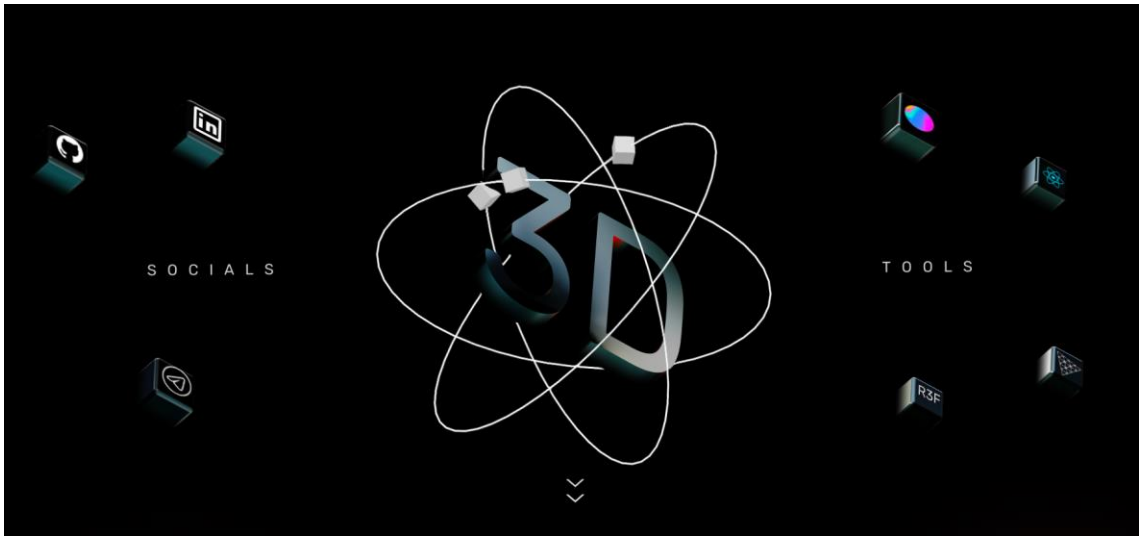


Figure 10. Intro section with revealed navigational menus


Figure 11. Response of functional button on mouse hover effect

Overall, this introductory section emphasizes interaction-driven design and visual responsiveness, inviting users to actively explore and interact with the 3D elements on screen. Through these combined features, the application provides a robust basis for understanding the interactive possibilities within the 3D interface integrated into the overall navigation structure, which makes the section visually engaging and functional.

### 4.2.2 Primitives Section

The Primitives section presents a structured interface divided into three areas: a menu for model selection and customization, a 3D model display, and a description area. Together, these segments provide users with an interactive workspace where they can explore various 3D primitives and modify their attributes and appearance in real-time.

The menu serves as the control center, listing available 3D primitives and allowing users to switch between them. When a list item is clicked, it moves to the top, revealing a settings button bound to the selected model's options tab. By clicking this settings button, users access an options panel that overlays the non-selected list items. This panel enables customization of the current model’s properties, such as size, amount of polygons, and functional variables, and offers a toggle option for displaying the model in wireframe mode. This interaction-focused design provides users with intuitive control over each 3D primitive, encouraging experimentation and adjustment.

The Visualisation of the 3D primitives is placed in the central section, showcasing the selected primitive with all applied settings. This area is rendered using React Three Fiber, allowing for variable-responsive and interactive visuals within the React framework. As users adjust settings in the menu, the displayed model immediately reflects these changes,

providing real-time feedback that enhances the UX. This section serves as the primary visual component of the interface, illustrating the customization and interactive possibilities of the application.

The description of the current primitive provides a concise description of the selected model. Each description explains the primitive's construction and potential applications, giving users context as they experiment with the model's attributes. Thereby, this section has the purpose of familiarizing users with the underlying structure and use cases of each primitive, promoting the exploration of 3D interfaces.

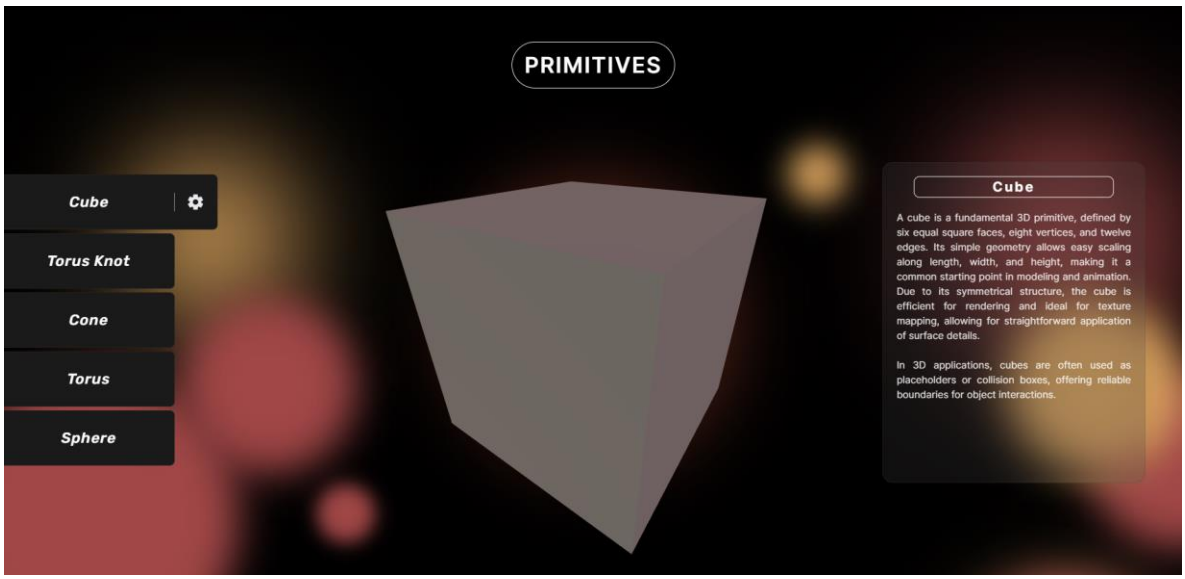


Figure 12. Primitives section

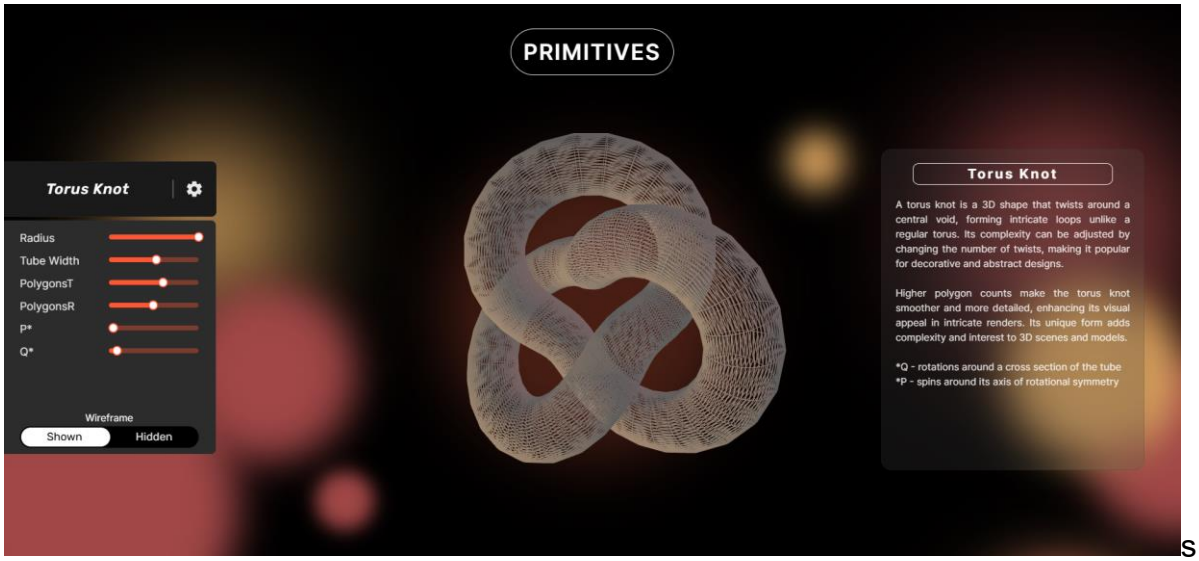


Figure 13. Primitives section with the opened options tab, Primitive's wireframe shown

This three-part interface allows for a hands-on approach to understanding 3D primitives, with a responsive display and intuitive settings that encourage users to actively engage with the models.

### 4.2.3 Shaders Section

The Shaders section provides an interactive setup where users can explore and customize shaders through a dynamic visualisation interface and accompanying shader descriptions. This section emphasises hands-on experience with real-time shader manipulation.

The visualization part features a 3D sphere model rendered using React Three Fiber (R3F), showcasing custom vertex and fragment shaders. Three animated vertex shaders are available for selection, each based on distinct noise algorithms, described in .glsl files: Simplex, Perlin, and an alternative Perlin variant. By clicking the sphere, users can cycle through these vertex shaders, observing how each algorithm uniquely distorts the sphere's geometry. Below the sphere, three colour selectors act as fragment shader options, allowing users to alter the sphere’s color by applying different fragment shaders. This arrangement lets users see how shader modifications affect both shape and colour, bringing out the power of vertex and fragment shaders in 3D rendering.

Two half-rounded sliders, powered by an external library, are placed on both sides of the sphere, each offering additional control over the visual effects. The left slider controls animation speed, enabling users to increase or decrease the velocity of the noise-based vertex animations. The right slider is employed to adjust the intensity of distortion, empowering users to amplify or moderate the impact of the vertex shader’s noise on the sphere's geometry. Jointly, these controls create a responsive and customizable environment for experimenting with shader parameters and appearance (Figure 14).

The descriptive part of this section provides an educational layer to accompany the interactive visualisation. It features two labelled boxes representing vertex and fragment shaders respectively. When clicked, the box expands into a text field that offers a concise explanation of the selected shader type. This functionality helps users familiarise themselves with the basic principles of both shaders, offering concise and focused insights (Figure 15).

By combining interactive controls with informative descriptions, the Shaders section creates a practical and engaging environment for understanding the basics of shader manipulation.

Users can directly engage with various shaders and parameters, enhancing their understanding of how vertex and fragment shaders influence 3D visualisations.

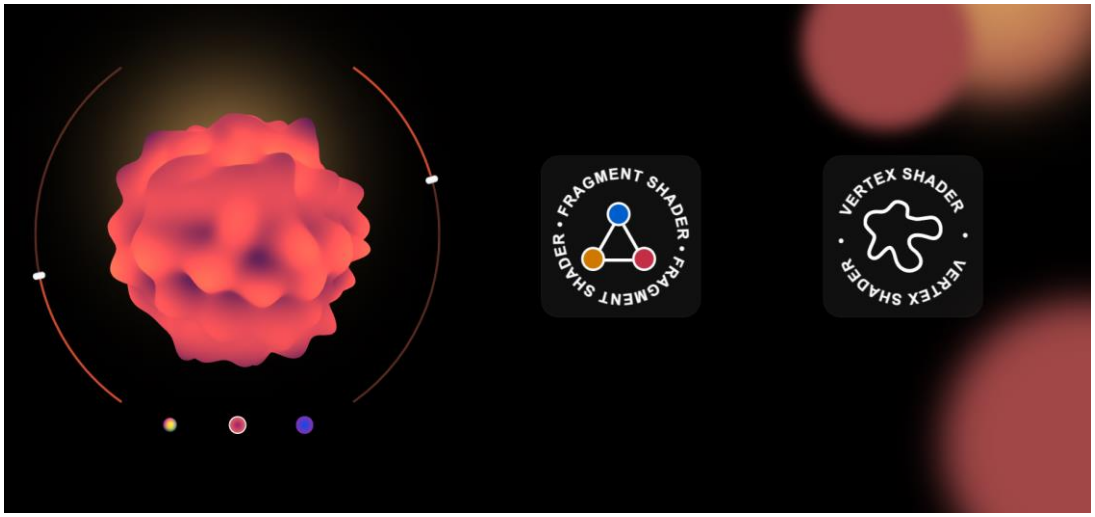


Figure 14. Shaders section

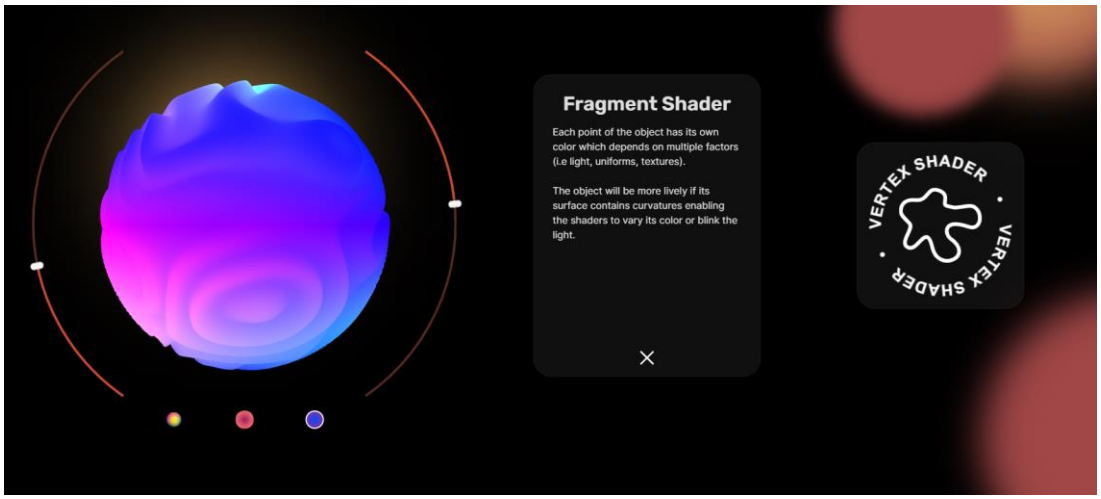


Figure 15. Shaders section, informational tab opened

### 4.2.4 Physics section

The **Physics** section of the application was crafted in Spline, focusing on realistic interactions through dynamic physics simulation. The scene showcases a pyramid-like stack of boxes, arranged in a card-house formation and illuminated by a single spotlight from above enhancing the realism of the environment and highlighting the stack's structure.

A key feature of this section is its interactivity, enabling users to drag, drop, and collide the boxes in real-time. Spline's built-in physics capabilities allow boxes to respond naturally to

user actions, transitioning and colliding realistically as they are manipulated, thereby transforming the scene into a responsive and tactile experience (Figures 16, 17).

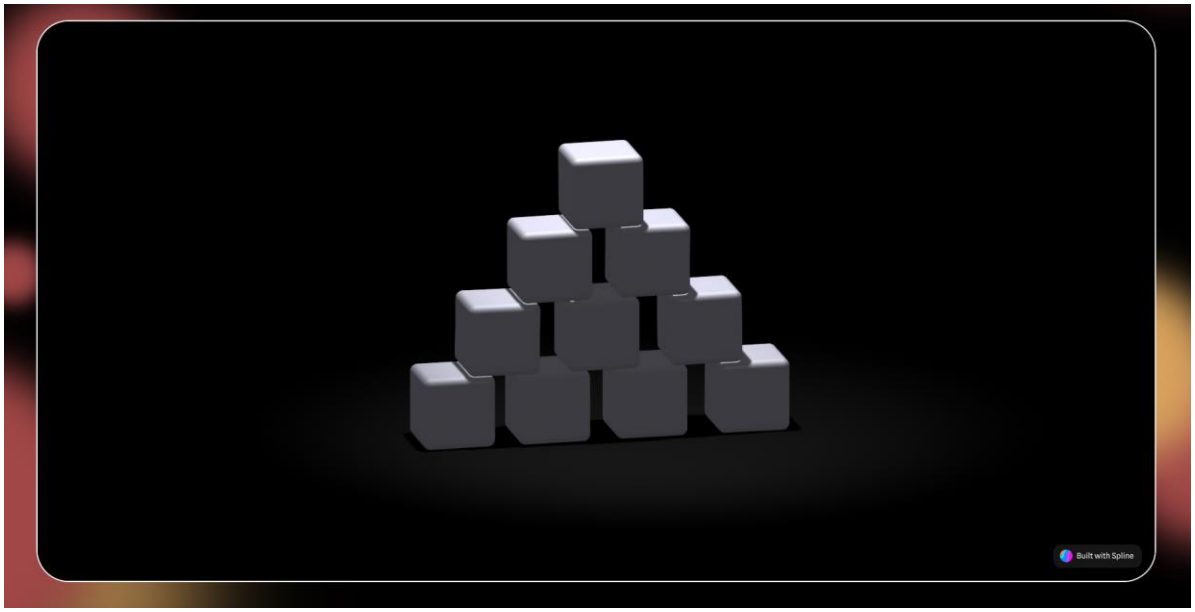


Figure 16. Physics section

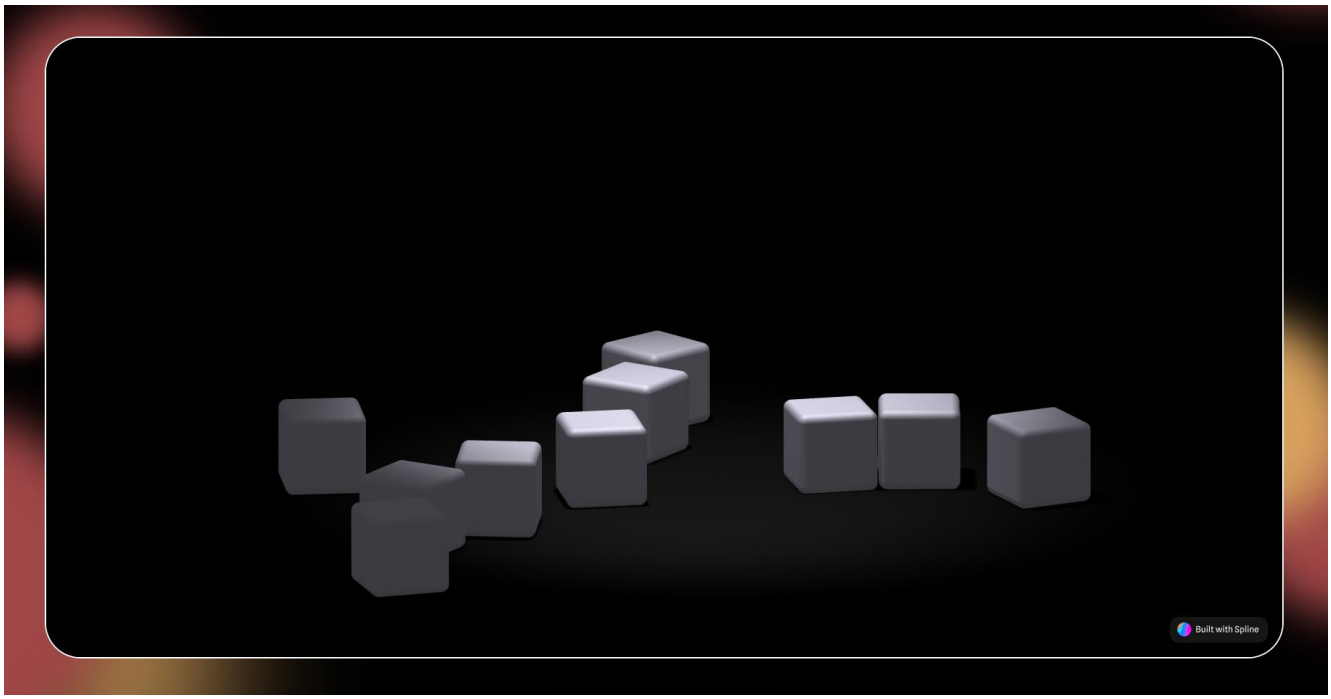


Figure 17. Physics section, physical response of blocks on interaction

By integrating Spline's physics tools, this scene demonstrates the potential of 3D interfaces to provide intuitive, hands-on interactions. Users can experiment with the stack, rearranging the boxes and testing the stability of the structure, bringing a dynamic layer to the 3D

experience. This section emphasizes the possibilities for realistic physics within web-based 3D applications, offering an engaging exploration of spatial dynamics.

### 4.2.5 User’s Models Integration

The Model Upload section was built using the R3F and provides an intuitive interface for users to import and view their custom 3D models of formats such as FBX, OBJ, GLTF and GLB or to upload files directly from their computer’s file explorer, making the feature accessible across various device configurations.

Once a model is uploaded, the upload interface becomes replaced by the display window, where the model is rendered for further examination or deletion using the reset button positioned on the model's right. R3F’s interactive capabilities enable users to inspect the model in 3D space by orbiting the camera around it. This functionality offers an extensive overview of a user's model, facilitating in-depth exploration from various perspectives and fostering an engaging experience for 3D content interaction.

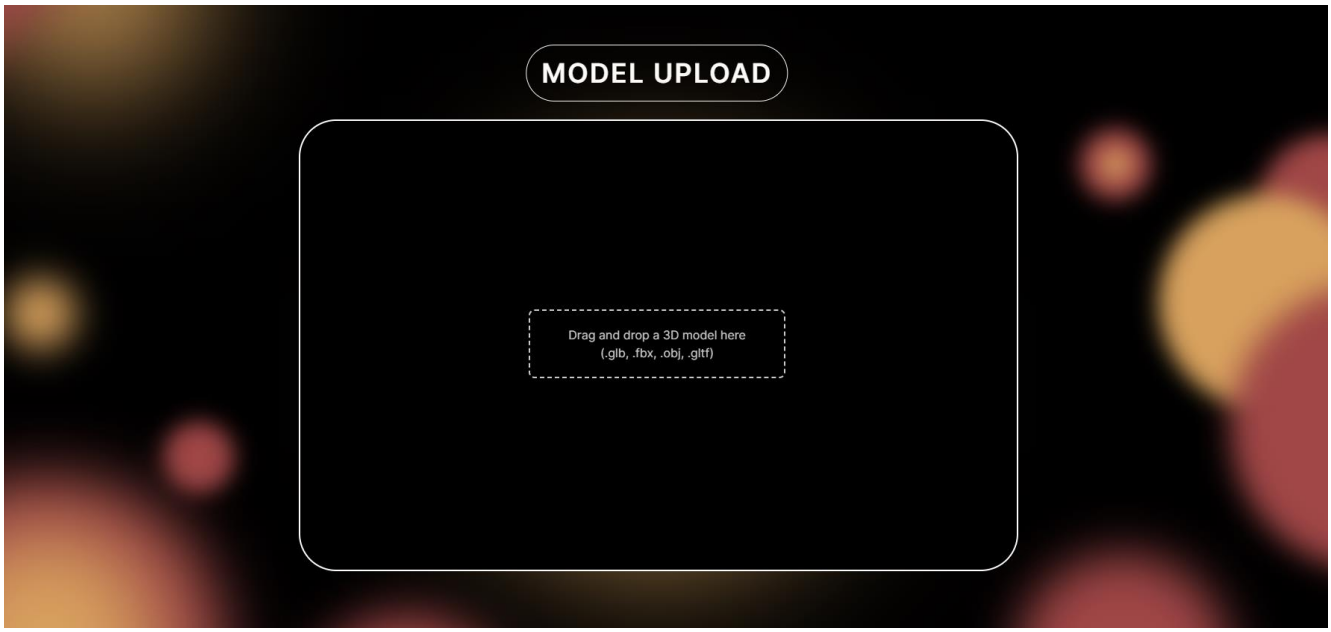


Figure 18. Model upload section

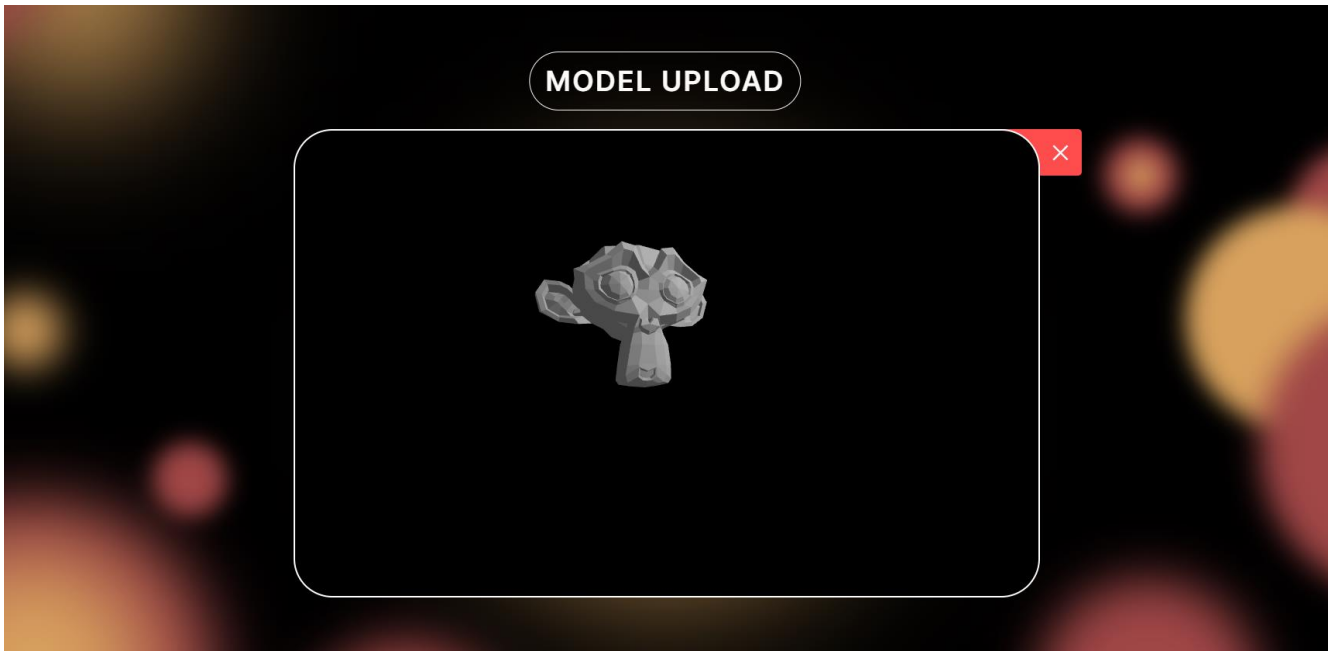


Figure 19. Model upload section with .glb model uploaded

# 5. CONCLUSION

At the outset of this thesis, several key expectations were set to guide the development of an interactive 3D web application. The primary goals were to create a React-based web application capable of integrating various implementations of interactive 3D models and interfaces, conduct a comparative analysis of 3D development tools and frameworks to contribute valuable insights into the strengths and limitations of different tools for 3D rendering and interaction within a React environment and provide an optimal user experience through the resulting application.

These goals and expectations were successfully achieved throughout the project. The resulting application showcases various interactive 3D elements within a React/Vite-based interface, which proved to be an effective combination, enabling rapid development. By employing a range of interactive 3D techniques and tools, the project presents diverse methods of embedding and manipulating 3D content within a web interface. Each section of the application explores unique aspects of 3D development—from animations and primitives to shader manipulation and physics simulation—highlighting the potential of integrating 3D into web applications with React and other supporting libraries. Thereby, the application serves as a comprehensive platform for exploring 3D content within a React framework, providing a predictable, reliable and interactive interface design within every section.

The project also followed the BEM methodology for CSS styling, which helped to maintain a modular and scalable CSS structure. By structuring the styles with BEM, the project achieved a clean separation of concerns, ensuring that each section of the application remains manageable and well-organised as additional features are implemented, or adjustments are applied.

Additionally, the project was deployed on GitHub Pages, making it readily accessible to users and providing a simple, no-cost solution for web hosting. GitHub Pages service supports static websites efficiently, and with Vite’s optimized build output, the deployment process was smooth, and the resulting website loads quickly for desktop users. This deployment strategy also makes the project easy to maintain and update, as new changes can be pushed directly from the VS Code Terminal.

This project also holds educational value, as it not only presents information on fundamental aspects of 3D development but also enables hands-on interaction with various models and

effects. The interactive nature of the application allows users to directly experiment with concepts discussed in the project, making it a valuable resource for students, developers, and enthusiasts interested in learning about 3D integration on the web.

### 5.1 Possible Improvements

While the current implementation meets initial objectives, there are several possible improvements that could elevate the application's usability and accessibility. First, a redesign focused on improving the user experience could enhance the overall interactivity and ease of navigation. Such a redesign might include adjustments to visual hierarchy, clearer indicators for interactive elements, and improved animations or transitions that guide the user through different sections intuitively. Making the interface more user-friendly could also improve the educational value of the application, particularly for those new to 3D web development concepts.

Another improvement area is expanding the application's compatibility for different screen resolutions, especially for mobile and tablet devices. While the current version is optimized for desktop screens, adapting the application to smaller screens would make it more accessible to a broader audience. This would involve optimizing layouts, adjusting 3D object interactions for touch inputs, and ensuring that animations and shaders perform well on lower-powered devices, which would increase the application's reach and usability.

Lastly, there is room for optimization regarding performance. Although the application runs smoothly in its current form, advanced techniques such as selective rendering, level of detail adjustments, and optimization of complex shader code could enhance performance further. This would be particularly beneficial for users with less powerful hardware, who might experience lag or slow loading times due to the 3D assets and animations.

These optimizations could make the app more efficient and robust, maintaining responsiveness even under heavier usage.

### 5.2 Complications During Development

The development of the introductory scene using Spline presented a unique set of challenges that demanded multiple iterations and refinements. The first iteration aimed for a visually rich

experience, involving numerous small 3D objects to create a detailed introductory scene. However, this design quickly hit a dead end, as the complexity of managing numerous tiny elements without a clear end animation resulted in performance inefficiencies and creative stagnation. This approach required assets to load in the background even when not visible to the user, which not only made the animation cluttered but also impacted performance, as there was no natural conclusion to the animation sequence.

In the second iteration, the scene attempted to streamline the assets by utilizing a single, heavier GLTF model, which was then multiplied as instances to create the desired environment. Despite the intention to reduce processing load through instancing, the model's size and complexity caused significant slowdowns and did not yield the expected performance improvements. Finally, the third iteration stripped the design down, minimizing the scene to a satisfactory and manageable level. This simplified version retained the core visual appeal without compromising performance, making it a more feasible solution for the introductory animation.

Another major complication arose during the development of the shader section, which was initially designed using vanilla Three.js. However, as the project grew, it became apparent that integrating Three.js directly within the React environment posed significant challenges, particularly in managing the DOM and object re-rendering. Every time a model's position or state was updated, a new instance of the model was created without effectively destroying the previous instance. This led to a buildup of invisible, yet CPU-intensive, models in the background, severely impacting performance. To resolve this, the project shifted to React Three Fiber (R3F), which is built specifically to manage Three.js within React's component-based architecture. R3F efficiently handled component re-rendering and object cleanup, providing a more optimized and React-friendly solution for integrating shaders and interactive 3D models without excessive CPU load.

### 5.3 Summary

In summary, this project successfully met its objectives, resulting in a React-based web application serving as an educational tool for 3D development concepts and practices. By comparing different 3D tools, demonstrating a variety of interactive techniques, and providing users with an interactive interface, the application fulfils its purpose as both a learning resource and a showcase of modern web-based 3D capabilities. Future iterations of this

project have the potential to further enhance its accessibility and user experience, broadening its utility as a resource for exploring and understanding the possibilities of 3D on the web. The result is a smooth, functional application that showcases essential 3D development concepts and provides an interactive, hands-on learning experience.

## BIBLIOGRAPHY

Aggarwal, S. (2018). Modern web-development using React.js. *International Journal of Recent Research Aspects, 5*(1), 133-137. https://ijrra.net/Vol5issue1/IJRRA-05-01-27.pdf

Babylon.js. (n.d). *Welcome to Babylon.js 7.0.* https://www.babylonjs.com/

Bielak, K., Borek, B., & Plechawska-Wójcik, M. (2022). Web application performance analysis using Angular, React and Vue.js frameworks. *Journal of Computer Sciences Institute, 23*, 77-83. https://doi.org/10.35784/jcsi.2827

Capała, Ł., & Skublewska-Paszkowska, M. (2018). Comparison of AngularJS and React.js frameworks based on a web application. *Journal of Computer Sciences Institute, 6, 82–86. https://doi.org/10.35784/jcsi.645*

Chen, S., Thaduri, U., & Ballamudi, V. (2019). Front-end development in react: an overview. *Engineering International, 7*(2), 117-126. https://doi.org/10.18034/ei.v7i2.662

Dimitrijević, M., Letić, J., & Obradovic, R. (2013) LIGHT AND SHADOW IN 3D MODELING. *Machine Design, 5*(3), 115-120. https://www.researchgate.net/publication/266316911_LIGHT_AND_SHADOW_IN_3D_MODELING

Eldridge, S. (n.d). *Cartesian coordinates*. Encyclopedia Britannica. https://www.britannica.com/science/Cartesian-coordinates

Encyclopedia Britannica. (2024a, October 10). *Spherical coordinate system*. https://www.britannica.com/science/spherical-coordinate-system

Encyclopedia Britannica. (2024b, August 22). *Polygon*. https://www.britannica.com/science/polygon-mathematics

Espinoza, J. (n.d). *3D Graphics: Fragment Shaders*. Medium. https://medium.com/@jrespinozah/3d-graphics-fragment-shaders-f50fe4a42da0

Evans, A., Romeo, M., Bahrehmand, A., Agenjo, J., & Blat, J. (2014). 3D graphics on the web: A survey. *Computers & graphics, 41*, 43-61. https://doi.org/10.1016/j.cag.2014.02.002

Gackenheimer, C. (2015). *Introduction to React.* Apress

GitHub Docs (n.d). *About GitHub Pages*. https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages

Johansson, J. (2021). *Performance and Ease of Use in 3D on the Web: Comparing Babylon. js with Three. js* (PA1445 Kandidatkurs i Programvaruteknik) [Bachelor's Thesis, Blekinge Institute of Technology]. DiVA. https://urn.kb.se/resolve?urn=urn%3Anbn%3Ase%3Abth-20977

Netlify (n.d). *Platform overview*. https://www.netlify.com/platform/

Npmjs (2024, October 10). *gh-pages. https://www.npmjs.com/package/gh-pages*

OpenGL Wiki. (2017, November 10). *Vertex Shader*. http://www.khronos.org/opengl/wiki_opengl/index.php?title=Vertex_Shader&oldid=14097

OpenGL Wiki. (2019, October 9). *Shader*. http://www.khronos.org/opengl/wiki_opengl/index.php?title=Shader&oldid=14593

OpenGL Wiki. (2020, November 25). *Fragment Shader*. http://www.khronos.org/opengl/wiki_opengl/index.php?title=Fragment_Shader&oldid=14712

React. (n.d.-a). *Your First Component*. https://react.dev/learn/your-first-component

React. (n.d.-b). *Passing Props to a Component*. https://react.dev/learn/passing-props-to-a-component

React. (n.d.-c). *State: A Component's Memory*. https://react.dev/learn/state-a-components-memory

React. (n.d.-d). *Built-in React Hooks*. https://react.dev/reference/react/hooks

React. (n.d.-e). *useState*. https://react.dev/reference/react/useState

React. (n.d.-f). *useEffect*. https://react.dev/reference/react/useEffect

React. (n.d.-g). *useContext*. https://react.dev/reference/react/useContext

React. (n.d.-h). *useReducer*. https://react.dev/reference/react/useReducer

React. (n.d.-i). *useMemo*. https://react.dev/reference/react/useMemo

React. (n.d.-j). *useCallback*. https://react.dev/reference/react/useCallback

React Three Fiber.docs. (n.d). *Introduction*. https://r3f.docs.pmnd.rs/getting-started/introduction#what-does-it-look-like?

Sandberg, E. (2019). *Creative Coding on the web in p5. js: a Library where JavaScript Meets Processing* (PA1445 Kandidatkurs i Programvaruteknik) [Bachelor's Thesis, Blekinge Institute of Technology]. DiVA. https://urn.kb.se/resolve?urn=urn%3Anbn%3Ase%3Abth-17941

Spline. (n.d). *Spline, a place to design and collaborate in 3D.* https://spline.design/

Thorne, E. (2024, August 26). *What is a normal in 3D modeling: Understanding the Concept of Normals in 3D Space*. Coohom. https://www.coohom.com/article/what-is-a-normal-in-3d-modeling?hl=ru

Three.js. (n.d.-a). *Hemisphere  Light.* https://threejs.org/docs/#api/en/lights/HemisphereLight)

Three.js. (n.d.-b). *Perspective Camera.* https://threejs.org/docs/#api/en/cameras/PerspectiveCamera

Three.js. (n.d.-c). *Orthographic Camera.* https://threejs.org/docs/#api/en/cameras/OrthographicCamera

Verma, V., & Walia, E. (2010). 3D Rendering-Techniques and challenges. *International Journal of Engineering and Technology*, 2(2), 29-33. https://www.researchgate.net/publication/50422357_3D_Rendering_-_Techniques_and_Challenges

Weisstein, E. (n.d.-a). *Cylindrical Coordinates*. Wolfram MathWorld. https://mathworld.wolfram.com/CylindricalCoordinates.html

Weisstein, E. (n.d.-b). *Vertex.* Wolfram MathWorld. https://mathworld.wolfram.com/Vertex.html

Wikipedia. (n.d). *Normal (geometry).* https://en.wikipedia.org/wiki/Normal_(geometry)

Wikipedia. (n.d). *UV Mapping*. https://en.wikipedia.org/wiki/UV_mapping