1.1 Introduction

This document presents the architecture and detailed design for the software for the Parametrix project. This project performs as a web application that can be used as a function-based animation suite that allows the user to quickly create animated objects that move, scale, and rotate according to mathematical equations.

1.1.1 System Objectives

The objective of this application is to provide an online interface to allow users, with sign in through Firebase authentication, to add objects to a scene, apply formulas to them, and have them animate on a certain axis of translation, rotation, or scale according to the formulas. This is done through the use of an object creation menu which instantiates the objects, and a formula menu by right clicking the object. The objects will be animated through a range of values according to a parametric variable, subdivided into a user-provided number of evenly-divided samples. The finished animation is then able to be saved as an animated GIF, with each sample being rendered as a single frame.

1.1.2 Hardware, Software, and Human Interfaces

The following Hardware, Software, and Human Interfaces are used in the design and development of Parametrix.

1.1.2.1 Hardware

The hardware used for the Parametrix project can be summarized as follows. Parametrix's development will be on a Dell laptop running Windows 10. This laptop is linked to either home or LMU servers to allow for development. This laptop also comes equipped with a basic keyboard and mouse pad for use with coding and developing. In addition, external peripheral mice or pen tablets may be hooked up to the device for easier mouse movement. Presentation of the project will be done on external devices such as projectors provided by LMU.

1.1.2.2 Software

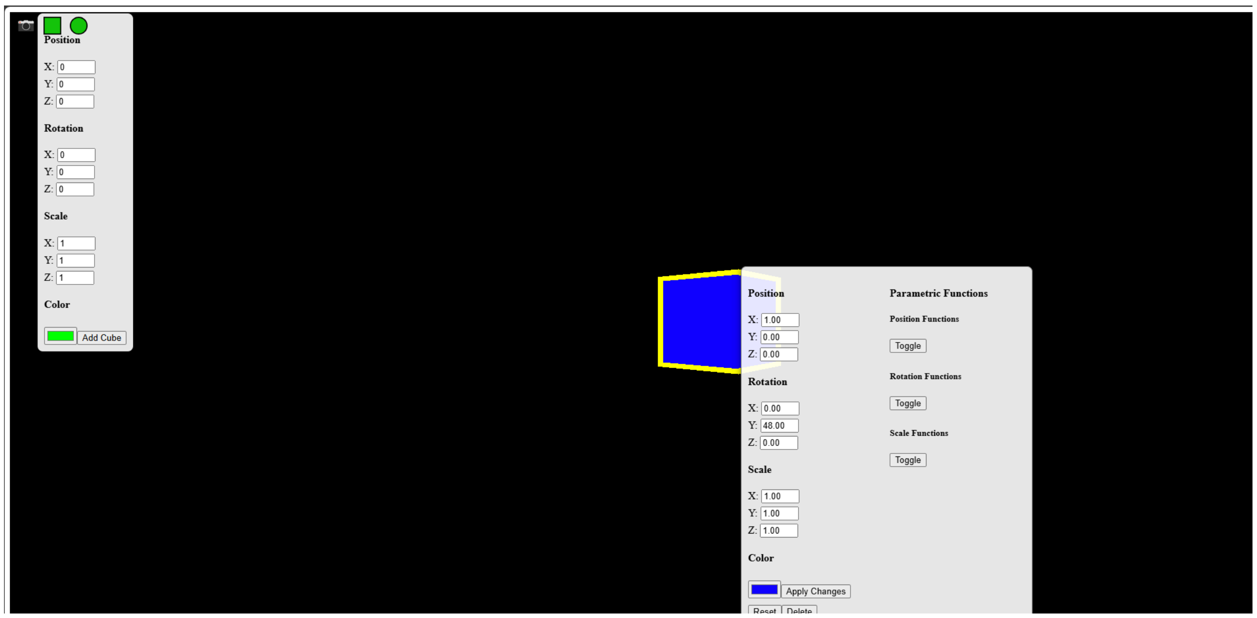
The software used for the Parametrix project can be summarized as follows. External browsers such as Safari, Google Chrome, and Firefox will be used to host the broadcast of the website to users when used. Visual Studio Code is the interface Parametrix files are created and developed in. React is used to develop in association with Node.js and Three.js to allow for the creation of the UI and code backend of Parametrix and its applications. Further parallelization of the created objects and data will be eventually managed by a worker-based system, with the possibility of Haskell in WebAssembly. Firebase is used with both authentication and storage to allow for easy sign in and storage for media with the Parametrix project. Firebase is also used to host the final website and provide a link for use.

1.1.2.3 User Interface

The UI for the Parametrix project can be summarized as follows. The UI of Parametrix is split into two main pages: the main editor homepage and the gallery page which will be described in two separate paragraphs.

1.1.2.3.1 Main Homepage

The main homepage of Parametrix will be presented as follows when first seen. The website will start with an empty editor screen with only some of the more basic tools available. The user will be able to add and manipulate objects like a signed in user would, albeit with more general functionality. The user will have a few options presented in the header: add objects from the object creation menu, click on the gallery button to move to the gallery page, or click on the sign in button to get started. Once signed in through Firebase, the user will get to interact with a new sign in button and media buttons to add to the whiteboard. The user can interact with all forms of media including the animation, right-click editor, gif export, mesh deformation, and camera tools to create their own project. Each tool will either open up a submenu for more details or will interact with the scene as is.



Position

X: 0
Y: 0
Z: 0

Rotation

X: 0
Y: 0
Z: 0

Scale

X: 1
Y: 1
Z: 1

Color

Add Cube

Position

X: 1.00
Y: 0.00
Z: 0.00

Rotation

X: 0.00
Y: 48.00
Z: 0.00

Scale

X: 1.00
Y: 1.00
Z: 1.00

Color

Apply Changes

Reset Delete

Parametric Functions

Position Functions

Toggle

Rotation Functions

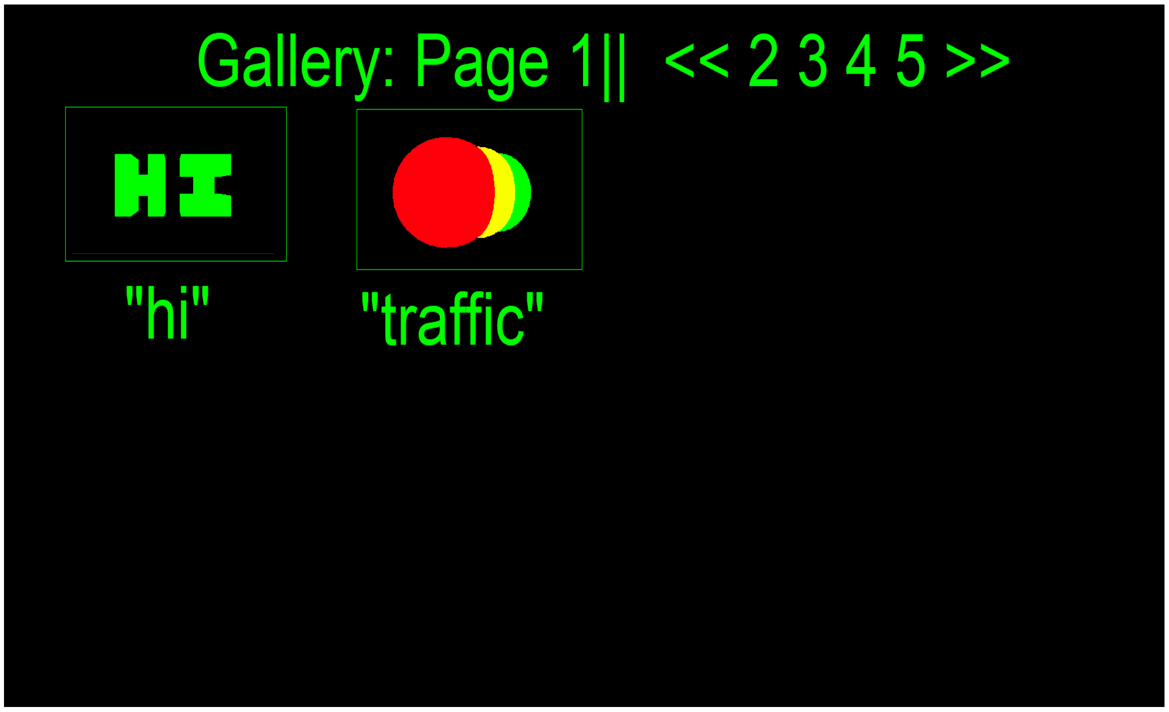
Toggle

Scale Functions

Toggle

1.1.2.3.2 Archive Page

The gallery page of Parametrix will be presented as follows when first seen. There will be a series of paginated thumbnails of uploaded GIF images. The user will have the ability to click on the thumbnails to view the animations in full and save them if they wish.



1.2 Architectural Design

The Architectural Design of Parametrix will be broken down into multiple different subsystems. These include the Animation Control, User Interface Control, and Database Control, described as the following:

Animation Control:

Parametrix's animation management system integrates a user-friendly UI with a robust toolset to handle the manipulation of various primitive 3D shapes. The interface includes an animation playback panel, a module to save projects and export them to GIFs, and the ability to modify and add functions to various elements, enabling streamlined management and interaction. On the backend, the database schema organizes GIFs, allowing for viewing and navigation, downloading, and uploading. This setup ensures Parametrix can store, categorize, and retrieve any project's export, supporting collaboration and creativity.

User Interface Control:

The UI of Parametrix is designed based on the main app.js file that displays the views for adding various objects, manipulating the camera controls, managing the states of all objects contiguously, and editing objects through the right-click menu. These are compiled with the css of the website to allow for a nice and presentable design. The parametricFunctions subclass hosts all of the possible functions that can be applied between the various dimensions of each object's scale, rotation, color, or translation. This, in particular, is managed by the animationManager class, which will simultaneously maintain the state of the scene's animation, keeping track of playback, sample rate, sampling interval, and scrubbing of the playback, while condensing all of these features into a UI displayed within the app. All of these files work together with the css, which makes sure the user interfaces remain streamlined, easy to navigate. The gallery.js class will allow for uploading from the main scene, as well as allowing for view into the gallery of user uploads.

Database Control:

The database of Parametrix is designed based on the main firebase_config.js file that allows for connection to the firebase database and allows for the set up of initialization, authentication, and storage for the project before exporting them as separate objects. These objects are used in two different ways. The authentication is used in the auth.js file that allows for sign in and sign out of Google accounts through a SignIn, SignOut, and useAuthentication functions. These link to loginButton.js where they are linked to button objects to allow for their interaction with the main homepage. The storage of the exported projects is maintained through the exportSceneToGIF file, which parses the finished scene frame by frame from the camera's perspective at each frame and layers each frame in a GIF image format. These finished GIFs can optionally be uploaded to the gallery, which is managed by gallery.js and uploaded to the online gallery page for presentation.

1.2.1 Major Components

Major subsystems present in Parametrix that correlate to the functional requirements of the software present in the Requirements Specification document can be described in the following:

- From 1.3.1.1-1.3.1.4, the User Interface subsystem is used to handle rendering the main webpage and its interactive menus. This is done primarily in the app.js file along with the cameraControls.js, menuTemplate.js and ui.js files and their associated css to host the normal website and allow for changes in presentation.
- From 1.3.1.5-1.3.1.11, the Object Management subsystem handles 3D object creation, deletion, and modification. It is mainly implemented through the objectManager.js, objects.js, objectEditMenu.js, and cubeMenu.js and sphereMenu.js files. They enable the user to create and modify basic 3D shapes with configurable position, rotation, scale, and color. The subsystem allows objects to be directly edited after creation through a right-click to open a context menu. This menu highlights the selected object with a yellow outline that turns blue when resetting and red when deleting the object.
- From 1.3.2.1-1.3.2.13, the Animation Control subsystem will handle the functional programming of animations. This functionality is implemented in animationManager.js, animationHud.js, and parametricFunctions.js. This subsystem enables the user to define animations using mathematical functions for position, rotation, scale, and color. It supports animations that loop indefinitely or run for a set duration. It provides a HUD interface for playback controls and looping toggles.

- 1.3.1.4, the Camera Control subsystem, handles user interaction with the scene camera. It is implemented in cameraControls.js. It allows the user to switch between perspective and orthographic camera views. It provides smooth transitions for camera position and rotations. It updates the camera's projection dynamically on window resize. In future implementations, it will be able to have its position and rotation functionally controlled.

- From 1.5.3-1.6.7, the Backend and Database subsystems handle firebase integration for user authentication and project storage. They are implemented through firebase methods in app.js, loginButton.js, auth.js, and gallery.js. This allows users to save projects and load projects from firebase storage, view projects on the gallery, and provide user authentication via Google sign-in.

1.2.2 Major Software Interactions

Major software interactions throughout the Parametrix application can be described by the following:

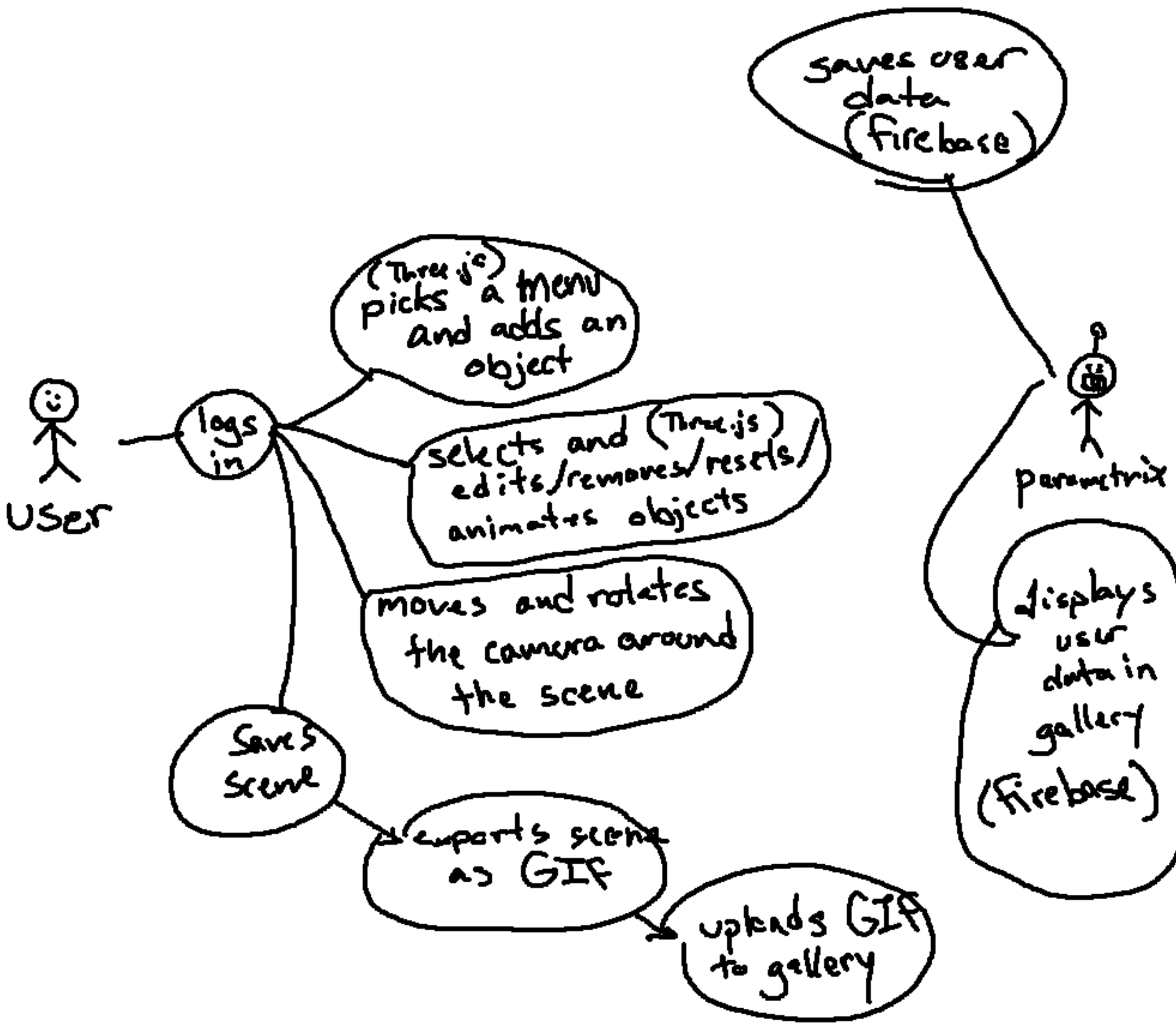
- Users can create 3D objects (cubes and spheres) via cubeMenu.js and sphereMenu.js. Objects are managed through objectManager.js, which stores and modifies them in the scene. Users can edit objects' position, rotation, scale, and color through the object edit menu (objectEditMenu.js).
- Users can assign parametric functions to control object transformations. Animations are stored in animationManager.js, which processes transformations over time. Users can toggle animation playback and looping using animationHud.js. Functions for defining animation behaviors are stored in parametricFunctions.js.
- The camera is initialized in cameraControls.js. Users can smoothly transition the camera's position and rotation. The camera supports toggling between orthographic and perspective views.

- Firebase authentication is used to manage user sessions. Users can save and load projects via Firebase cloud storage.

1.2.3 Architectural Design Diagrams

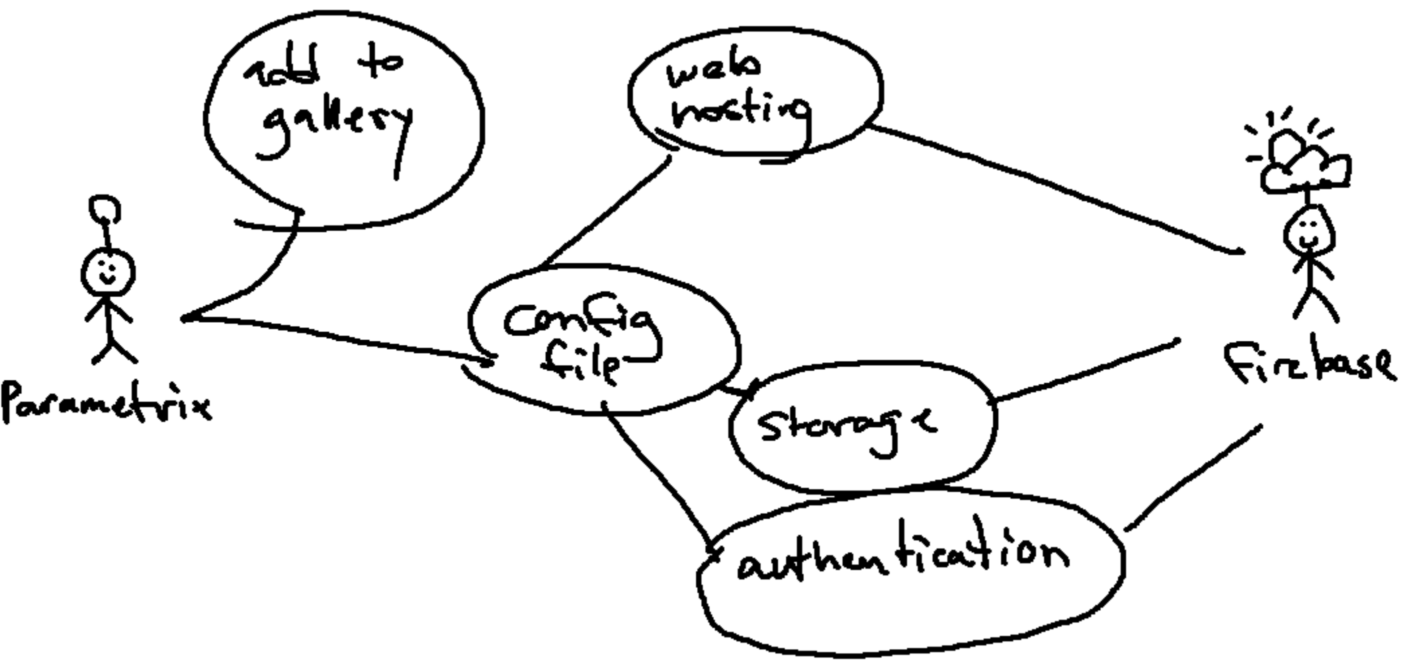
Here are some diagrams for Parametrix applications at an architectural level:

Overall System



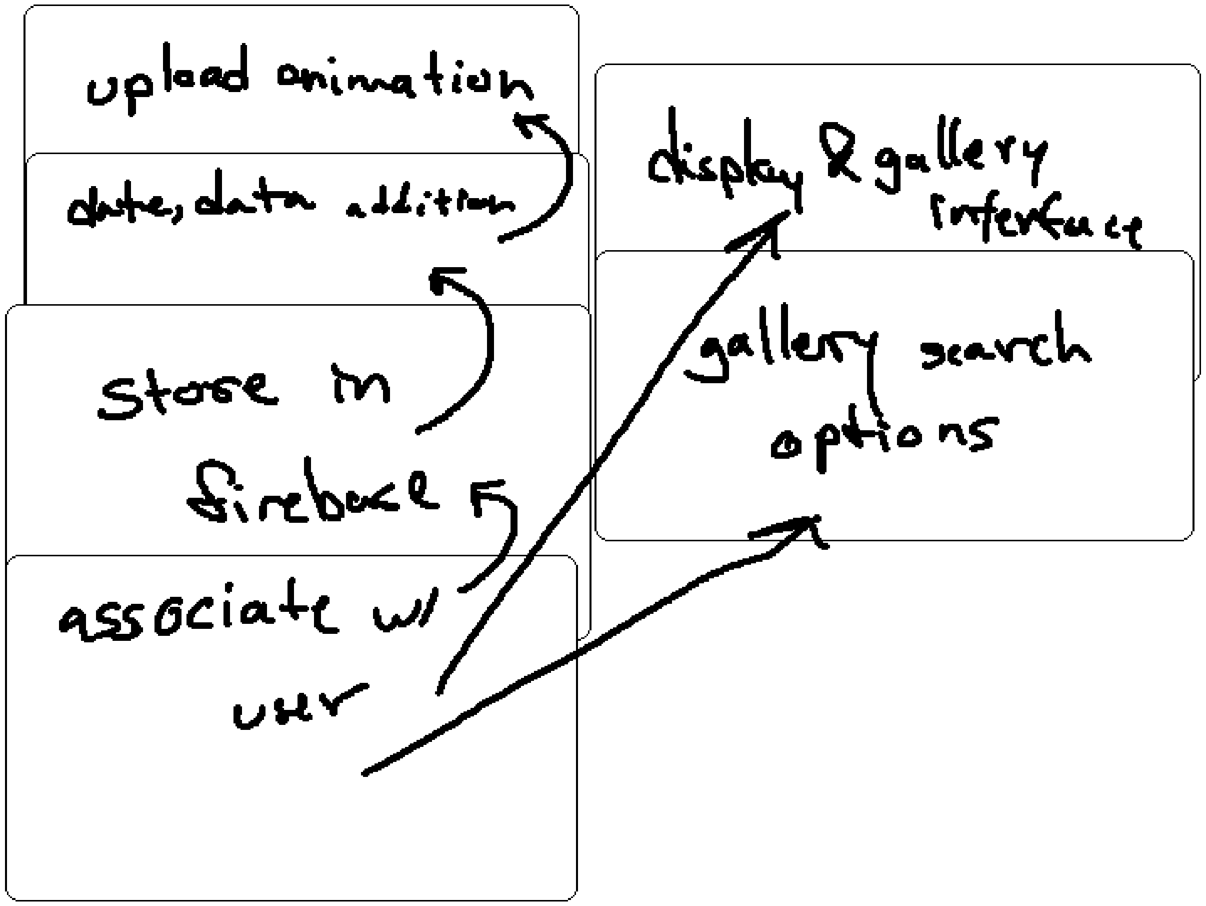
User
logs in
(Three.js) picks a menu and adds an object
selects and (Three.js) edits/removes/resets/animates objects
moves and rotates the camera around the scene
saves scene
exports scene as GIF
uploads GIF to gallery
saves user data (firebase)
parametrix
displays user data in gallery (firebase)

Firebase Integration:



Media Interaction Chart:

User Media Interaction Flow in Parametrix



1.3 CSC and CSU Descriptions

The Detailed Design of Parametrix will be broken down into multiple CSC's and CSU's in accordance with the system. These will be highlighted down below:

<u>App and Header Presentator</u>

The CSUs involved in this section of Parametrix include:

- App
- Camera Controls
- Object Manager

● Objects (detailed in archive section)

● Object Edit Menu (detailed in media interactions section)

● Sphere Menu

● Cube Menu

● Google Sign-In Container

● Animation Manager

● Animation HUD

● Parametric Functions

● UI

● Menu Template

Gallery Presentator

The CSUs involved in this section of Parametrix include:

● Gallery

● Export Scene to GIF

Authorization and Storage useEffects and Providers

● Google Sign-In/Sign-out Container

1.3.1 Class Descriptions

The following sections provide the details of all the classes used in the Parametrix application. These will be split into subsections which will be shown here:

1.3.1.1 Menu Template

The menuTemplate.js class provides a base structure for interactive UI elements within Parametrix. This class defines a reusable menu component that is dynamically positioned within the scene. These elements are customizable, allowing for modularity. Different emojis can be used to represent various tools and objects that can have their menus represented, and can be easily moved around the page if edits to the UI need to be made.

Fields:

- menuContainer (HTMLElement): A div element representing the menu's container, styled with a semi-transparent background.
- toggleButton (HTMLElement): A button element that toggles the menu's visibility.

Methods:

- createMenuTemplate(emoji, top, left): Creates and positions the menu and toggle button within the UI. The menu starts hidden and is revealed when the button is clicked.

1.3.1.2 UI

This class involves handling how the UI is rendered. It makes a container that makes sure the menus are kept within the dimensions of the page, such as the camera and buttons for adding objects. It uses a flex-style css to constrain the menus within visible screen space.

Fields:

- container (HTMLElement): The main UI container which holds the controls.

Methods:

- createUI(): Creates a flex-based UI layout positioned at the bottom left of the screen.

1.3.1.3 Objects

This class is meant to encapsulate various methods for the creation of several different basic shapes in Three.js. The shapes are meant to be quickly-assembled readymades that can be scaled up for more complicated code and used consistently throughout the source without having to hard-code a new mesh and normal material every time.

Methods:

- createCube(): Creates a cube with a given mesh.
- createSphere(): Creates a sphere with a given mesh.

1.3.1.4 Object Manager

The object manager is a class that will take care of pushing and disposing of objects created in the scene. It will contain the array of all objects that are to be collectively pushed to the scene, animated on, and disposed of properly without leaking memory. It contains individual methods for adding an object, removing an object, and containing the objects.

Fields:

- objects (Array): stores all managed Three.js objects in the scene, and allows tracking of added objects for easy retrieval or removal.

Methods:

- addObject(scene, object): adds a Three.js mesh to the scene, stores the objects in the objects array.
- getObjects(): returns the array of managed objects, and allows external access to the current list of the scene's objects.
- removeObject(scene, object): removes a specified object from the scene, deleting its

geometry and material and disposing of it from the objects array.

1.3.1.5 Cube and Sphere Menus

These menus are emoji icons of a sphere and a cube that when clicked allow for the creation of a cube with user-defined X Y and Z dimensions for position, scale, and rotation, as well as RGB values for a color. The menus for each object are nearly identical, with the only difference being whichever kind of object is being added.

Fields:

- menuContainer (HTMLElement): A div element representing the menu's container, styled with a semi-transparent background.

Methods:

- createCubeMenu(scene): Adds a menu which shows a dialog for adding a cube of user-defined position, scale, rotation, and color to the scene.
- createSphereMenu(scene): Adds a menu which shows a dialog for adding a sphere of user-defined position, scale, rotation, and color to the scene.

1.3.1.6 Object Edit Menu

This class is responsible for the menu that appears when an object in the scene is right-clicked. It allows users to edit the properties of an existing object, reset the object to program defaults, or delete the object from the scene. It also highlights the selected item in a yellow outline, which changes to blue when the mouse hovers over the reset button, and red when the mouse hovers over the delete button. The apply button, when clicked, applies the changes in the menu to the selected object.

1.3.1.7 Camera Controls

This is a class that allows the user to open a camera menu by clicking a camera icon, and use the menu to control the position and rotation of the camera. The camera may allow for animation of the position and rotation. The inputted positions the user types in will be moved and rotated to with quadratic easing, which can allow for distinction between multiple rotations, or covering large distances within the scene.

1.3.1.8 Animation Manager

This class will contain the bulk of methods for controlling how objects animate every frame within the scene. Its methods may be implemented into the Object Manager to allow for controlled animation of the objects array. The animations will maintain a user-defined period the animation cycles through, either once or in an infinite loop, with a user-defined number of samples which will be moved through as individual frames of the animation.

1.3.1.9 Animation HUD

This class allows the heads-up display of the animation manager to be seen, which allows for direct real-time manipulation of the scene’s animation. The objects in the scene will update according to changes in scrubbing, sample rate, period, as well as play/pause functionality.

1.3.1.10 Parametric Functions

This class contains various types of functions, ranging from simpler linear functions to more complicated sinusoidal and quadratic functions. The class defines the variables that govern each, and will export its variables to the Object Edit Menu to allow for easy editing. The user-defined values will then be used to generate the samples that govern the animations that reference them.

1.3.1.11 App

The App.js file is the main entry point for Parametrix, linking all subsystems together.

Fields:

- Scene (THREE.Scene): the primary rendering container.
- Renderer (THREE.Renderer): handles graphics rendering.
- CameraControls (CameraControls): The active camera controller.

Methods:

- initializeScene(): Sets up the Three.js environment.
- renderLoop(): Continuously updates animations and objects.

1.3.2 Interface Descriptions

The following sections provide the details of all the interfaces used in the Parametrix application. These will be split into subsections which will be shown here:

App and Gallery Presentator

- **React (UseState):**

React (UseState): Enables state management across the App and Menu files. This is used to track object states, manage animations, and modify object properties. Additionally, this interface includes functionality for exporting animations to GIF format or saving a project file that Parametrix can load and interpret.

- **Gallery:**

The gallery component allows users to access saved animation projects. It provides navigation controls through an onBack() function, which enables users to return to the main workspace. Additionally, it integrates functions such as handleGalleryClick and setShowGallery to control visibility.

- **Header:**

The Header component manages UI elements such as the sign-in and sign-out functionality, navigation between views, and settings. It interacts with Firebase to authenticate users and retrieve relevant user data.

- **React (useEffect):**

Provides lifecycle control for updating animations, synchronizing UI elements, and handling interactions that depend on state changes. This ensures that real-time updates occur efficiently across the application.

- **SignIn**

The SignIn component is a small interaction used for displaying the sign in container that is provided with google firebase.

- **SignOut**

The SignOut component is a small interaction used for displaying the sign out container that is provided with google firebase.

- **UseAuthentication:**

Works with Firebase to verify if a user is authorized within the application. It updates user data accordingly and manages session persistence.

Gallery Presentator

- **Firebase (getStorage, ref, listAll, getDownloadURL)**

The getStorage component is used in association with the const storage in order to gain access to the storage in the Firebase database. This also gives access to the current rules set in the Firebase database which will be applied to the project when used. Ref is a simple reference to the storage animations and animationThumbnails folder allowing it to be modified in the code. listAll is used to list all of the current screenshots in both the project and the Firebase database for comparison. These are used with the const results to check if the data between apps is successfully updated and current.

- **React (useEffect, useState)**

Both of these React elements are described earlier in documentation.

1.3.3 Data Structure Descriptions

The following sections provide the details of all the data structures used in the Parametrix application. These will be split into subsections which will be shown here:

Arrays: In Parametrix, arrays play a crucial role in managing and organizing objects, animations, and interactions. Arrays such as objects, animations, store general complete states of every object's state.

Objects: Objects in Parametrix encapsulate all relevant properties for 3D elements, animations, and user interactions. Each object—such as a cube or sphere, contains attributes defining its type, position, rotation, scale, material properties, and animation state. This structure allows seamless transformations and enables users to programmatically control object behavior. Additionally, interaction states such as hoveredObject and selectedObject are stored as objects to track real-time user interactions.

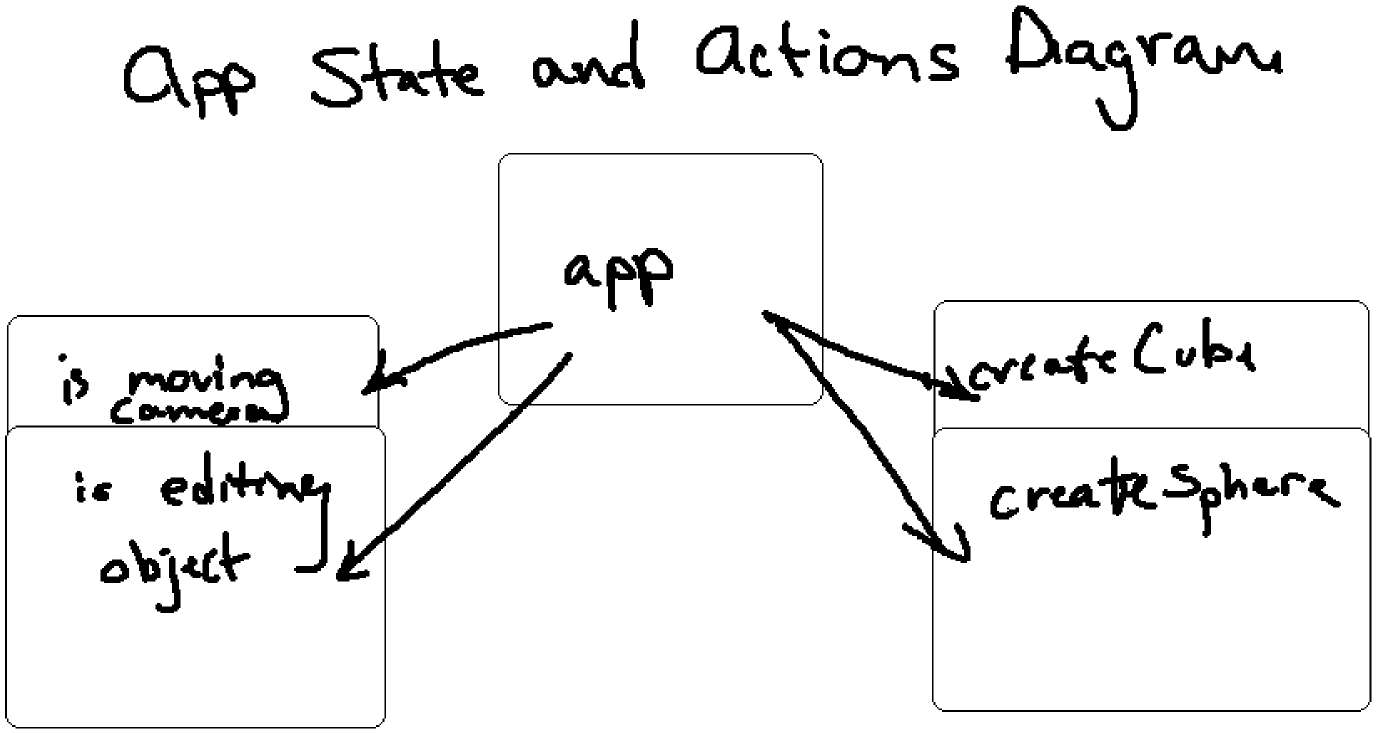
References: References (using React's useRef hook) are critical for preserving state between renders without triggering re-renders. Examples include cameraRef (storing the current camera instance), selectedObjectRef (tracking the currently selected 3D element), and animationLoopRef (maintaining a reference to active animation loops). These references improve performance by avoiding unnecessary recalculations while ensuring state consistency across complex user interactions like dragging, resizing, and animation playback.

Booleans: Boolean variables in Parametrix help manage active/inactive states for tools, animations, and object selections. Variables such as isAnimating, isTransforming, and isDragging track whether an animation is running, an object is being edited, or the user is interacting with an object. These booleans ensure that only one interaction state is active at a time, preventing conflicts between user inputs and system responses. They enable precise control over UI behavior, ensuring smooth transitions between different modes such as object editing, animation playback, and camera manipulation.

1.3.4 Design Diagrams

Here are some diagrams for Parametrix CSCs and CSUs at a design level:

App State



1.4 Database Design and Description

Parametrix will use Google Firebase to access database based applications including hosting, authentication, and storage. All of the four team members have access to the Parametrix Firebase console. Parametrix hosting is done by using the firebase servers to host the website. The terminal is used to run updates and push changes to the online website. Authentication is currently only used with Google accounts that can be logged into the database, giving information on a user's email and sign in date for possible moderation purposes. Storage will be divided into subsections including but not limited to:

Screenshots:

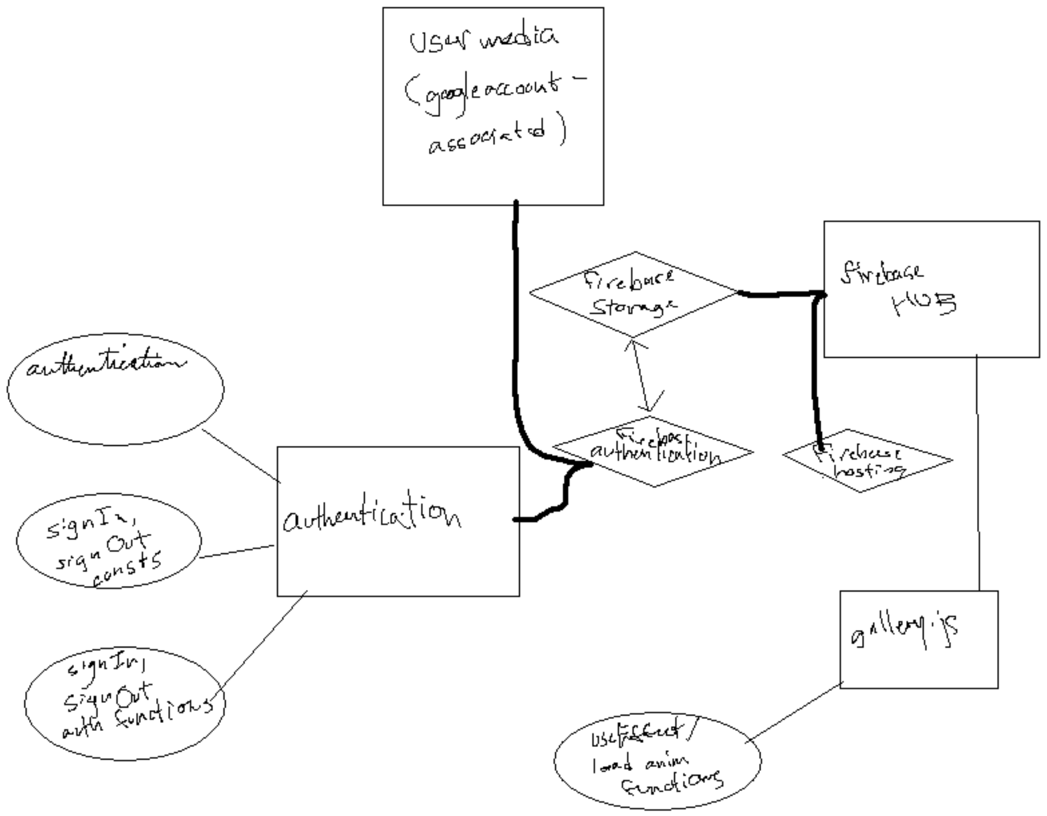
Timed screenshots of the websites that upload to Firebase every three days through the puppeteer api. Uploaded back onto the Archive page for archival purposes.

Media:

User associated media objects added to the whiteboard and stored in the database through the use of Text, Images, Gifs, and Drawings. Can be deleted from storage if violating moderation rules thereby removing it from the whiteboard public website.

1.4.1 Database Design ER Diagram

Here is a diagram describing Parametrix's use of Firebase through its systems:



1.4.2 Database Access

The database will be accessed through several avenues throughout the Parametrix code including the following:

ConfigFirebase.js:

Pulls from the personal Parametrix project website and allows for access of functions such as getAuth, getStorage, and initializeApp before exporting them as objects titled auth, storage, and app.

Gallery.js:

Uses getStorage from the config file to create a const titled “storage”. Works in tandem with a “useEffect” function to load screenshots from storage by using the previously mentioned “galleryAnimRef” to list all of the images. The data is then displayed in association with the “galleryAnimData” function to link dates provided in the Firebase system to animations shown. The images are finally displayed in the archive with the function “loadAnimations” and a HTML class known as “gallery-grid” working together with css.

App.js

Imports the “useAuthentication” function from ConfigFirebase.js to link to a const titled “user.” Creates references titled “onSignIn” and “onSignOut” that are utilized with the Sign in and Sign out functions and used in Header.js.

1.4.3 Database Security

To ensure security from the Firebase platform, Firebase allows for specific rules based for storage and authentication. For authentication, only Google users are allowed to sign into the website. As mentioned, Firebase gives access to the users account information including email and sign in date. Firebase also gives access to the user UID which can allow for moderation to ban or limit a user from the Parametrix website if needed. For storage, Firebase allows for special security rules to be enabled or disabled to prevent or allow user activities to happen. Currently the rules allow for users to interact with the whiteboard sign in with each media piece they add being associated with their account. Users are not allowed to interact with whiteboard storage and other users' media seen on the whiteboard preventing errors. On our end, we are looking to establish rules and checks in the code that will prevent users from messing with settings that Firebase cannot block out.