Table of Contents

- 1 Project DIGI-MODE Masser - Technical documentation
  - 1.1 What is massing?
  - 1.2 The problem
  - 1.3 The idea behind the Project DIGI-MODE Masser
  - 1.4 About the documentation
- 2 Server setup
  - 2.1 Dependencies
  - 2.2 Server installation
  - 2.3 Starting the server
  - 2.4 Password protection
  - 2.5 Documentation
- 3 User guide
  - 3.1 Connecting
  - 3.2 What is a room?
  - 3.3 Lobby
    - 3.3.1 Joining a room
    - 3.3.2 Creating a room
  - 3.4 Custom model format
  - 3.5 Inside the room
    - 3.5.1 Camera controls
    - 3.5.2 Creating and editing masses
    - 3.5.3 Cooperation and presenter mode
    - 3.5.4 Virtual reality
- 4 Frameworks and web technologies
  - 4.1 HTML5 and Javascript
  - 4.2 What is an API?
  - 4.3 WebGL
  - 4.4 WebXR
  - 4.5 Three.js
  - 4.6 Node.js
  - 4.7 Entity system (Dnz Framework)
- 5 Assets and tools
  - 5.1 3D scanning
  - 5.2 MakeHuman
  - 5.3 3D editor
  - 5.4 Sound repositories and editor
  - 5.5 Source-code editor
- 6 Development
  - 6.1 Deciding on a game engine/3D framework
  - 6.2 Project structure
  - 6.3 Agile software development
  - 6.4 The first sprint

- 6.5 The second sprint
- 6.6 The third sprint
- 6.7 The fourth sprint
- 6.8 Retrospective meetings and UX testing

1 Project DIGI-MODE Masser - Technical documentation

Leverage from the EU 2014–2020

European Union
European Regional Development Fund

Regional Council of Ostrobothnia

1.1 What is massing?

The architectural design process begins by studying massing. Massing is a term in architecture which refers to the study of buildings as three-dimensional shapes and volumes. In massing buildings are treated as compositions of basic shapes such as boxes, pyramids and gables. A complex shape is a composition of multiple simple shapes added or subtracted together. For example, a typical house can be seen as a rectangular box with a triangular roof; and a skyscraper can be seen as a simple box.

Massing is useful when studying the aesthetics and visual profile of a new buildings. Massing can also be used when calculating the surface area and energy expenditure caused by heating.

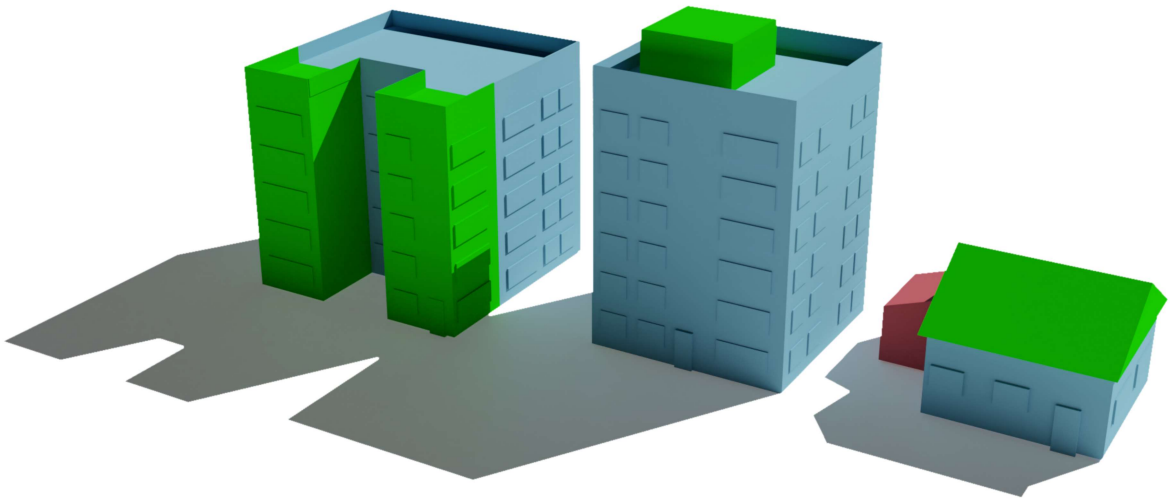


Figure 1: Massing deals with the composite shapes of buildings

1.2 The problem

The customer correspondence during the massing phase can be a long and complex process. The architects apply incremental changes to the design which are then shown to the customer, who in turn may suggest even more changes to add. There's a lot of communication between the architects and the customer as the design evolves. Small changes to the buildings necessitate an update the visual model by the architects, and subsequently future meetings with the customer. This may happen many times, and this back and forth can slow down the overall project significantly.

With that in mind, architects need:

- A way to easily make changes to a construction project
- An easy and quick method of collaboration between architects and customer
- A way of visualizing building masses and how they fit into the environment (visual profile)

The solution could be to use a tool which enables quick and immediate collaboration between the architects and the customer. Something like an online meeting for massing, in which the architect could edit the design in real-time together with the customer and other architects.

1.3 The idea behind the Project DIGI-MODE Masser

The Project DIGI-MODE Masser is a digital collaboration tool developed in conjunction with the local architectural company NAC Arkkitehdit Oy. The tool is intended to be used by architects in collaboration with customers (or by architects alone) to quickly design, preview and present building projects inside a realistic three-dimensional environment.

The Masser displays a three-dimensional replica of the real-world environment surrounding the preselected construction zone. Multiple architects and spectators can join and collaborate within the same environment.

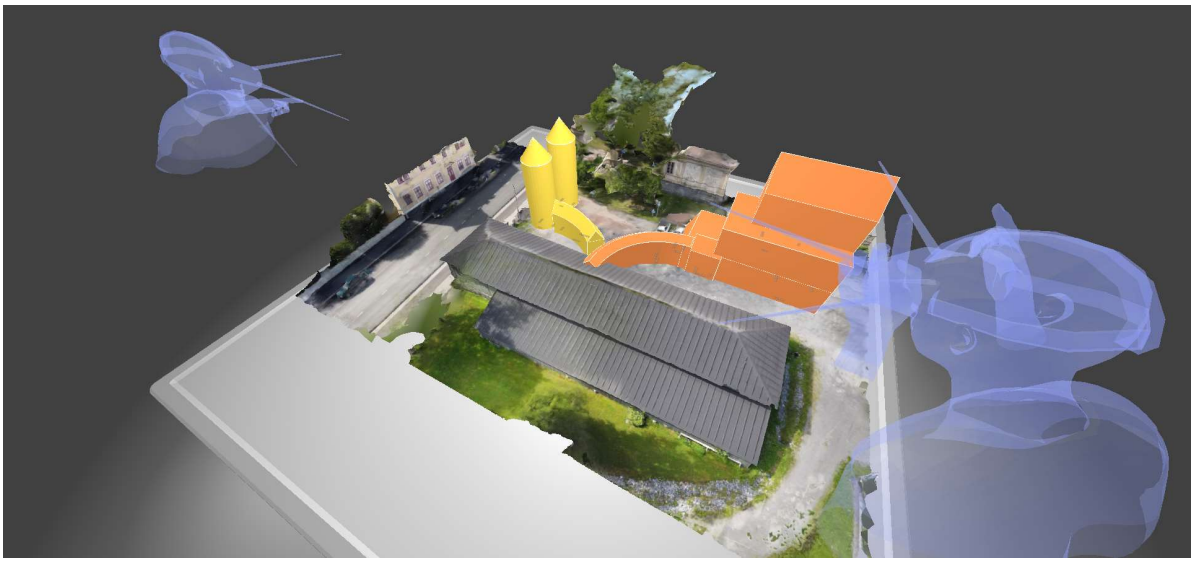


Figure 2: Desktop users can collaborate with VR users

The architect can add different building masses on top of the real-world ground, or on top of existing buildings. The masses can be edited to be of a different size, shape and color. These masses have a wide array of settings to allow them to be shaped into close approximations of the planned buildings.

The Masser has many other features:

- **Room system and lobby**: The Masser server can host any number of separate rooms
- **Multi-user collaboration**: Multiple users can join and collaborate inside the same room
- **Per-user permissions**: Individual users can be allowed to edit masses
- **Presenter mode**: The view of individual users can be set to follow a presenter's view
- **First person view**: The environment can be viewed from a ground perspective
- **Mass measurements**: Masses have measurement indicators for size and area
- **Complex masses**: Masses can have different shapes, colors, curvature and height
- **Multi-floor masses**: Masses can have multiple unique floors
- **VR-support**: The Masser can be run in VR for a more immersive experience
- **Smartphone support**: The Masser runs on iPhone/Android/Windows Phone

- **Saving and loading**: Masses can be saved and loaded from file

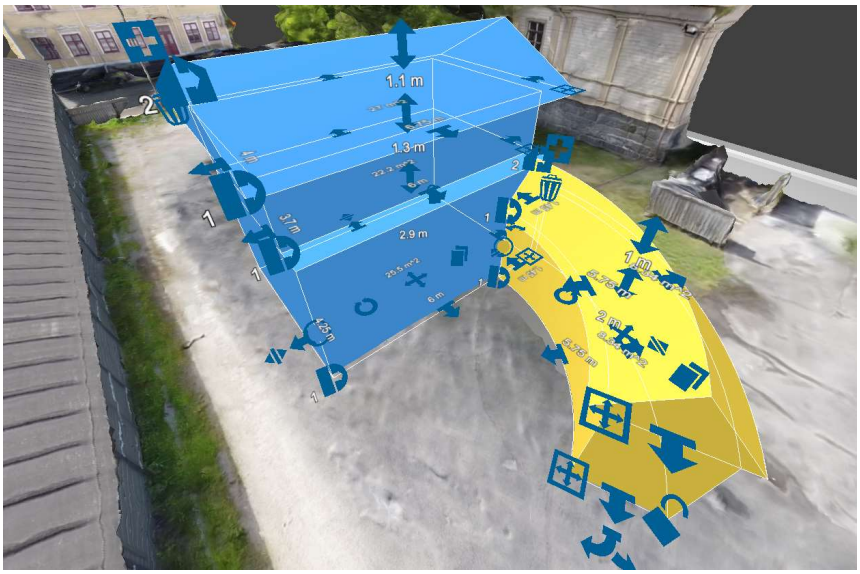


Figure 3: A complex composite mass

The masser has a sophisticated control system which allows for easy moving, scaling, rotating, mirroring and cloning of masses. Finished masses can be locked to reduce clutter.

1.4 About the documentation

See the Server setup chapter for instructions on how to prepare the Masser server. The User guide chapter contains the end-user instructions on how to control and use the Masser tool. The Frameworks and web technologies chapter gives a brief introduction into the different technologies used. The Assets and tools chapter describes the asset repositories and tools used to create content for the Masser. Finally, the Development chapter details the development process.

2 Server setup

2.1 Dependencies

The Masser is a web application based on HTML5, thus clients running the application only need an up-to-date web browser and the address to the server hosting the Masser.

The server hosting the Masser is based on Node.js. Node.js is an open-source back-end Javascript engine often used to host web servers. Node.js applications use the Node package manager to install dependencies.

2.2 Server installation

The Node.js server can be installed locally or on a remote server or cloud platform (such as Heroku or aws).

The `/app` directory contains both the Node.js server and the publicly hosted static files inside `/app/static`. Inside the `/app` directory, execute the following command to install the server dependencies:

```
npm install
```

2.3 Starting the server

After successfully installing the application dependencies, the server can be started using:

```
npm start
```

By default, the server listens on all the computer's network interfaces. The Masser app can be opened at http://public-ip-here:8080 on the network or http://127.0.0.1:8080 on the local computer. An SSL (HTTPS) certificate is required for VR to work.

2.4 Password protection

The Masser server is password protected. The user passwords can be changed in `/app/index.js` in the `users` variable.

2.5 Documentation

This documentation was written in markdown and exported using MkDocs. The `/app/documentation` directory tree contains the documentation files. To export the documentation run:

```
npm run docs
```

The documentation will be exported to the `/app/static/documentation` directory.

3 User guide

3.1 Connecting

The Masser tool can be opened in a compatible web browser by entering to the server URL in the address bar. By default, the server is password protected. Users wanting to open the tool need to get the login details from the server administrator.

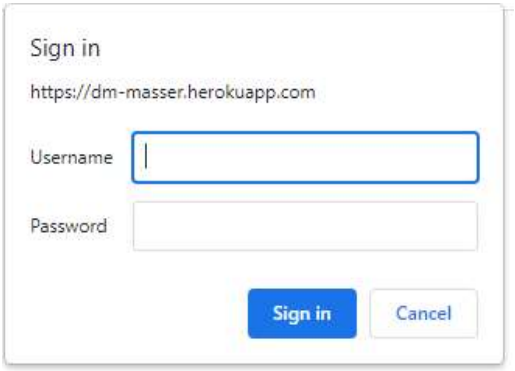


Figure 4: Standard browser log-in screen

3.2 What is a room?

The Masser server is capable of hosting one or more sessions or "rooms". Rooms have their own users, 3D environment and building masses. Users in one room can't see users in another room.

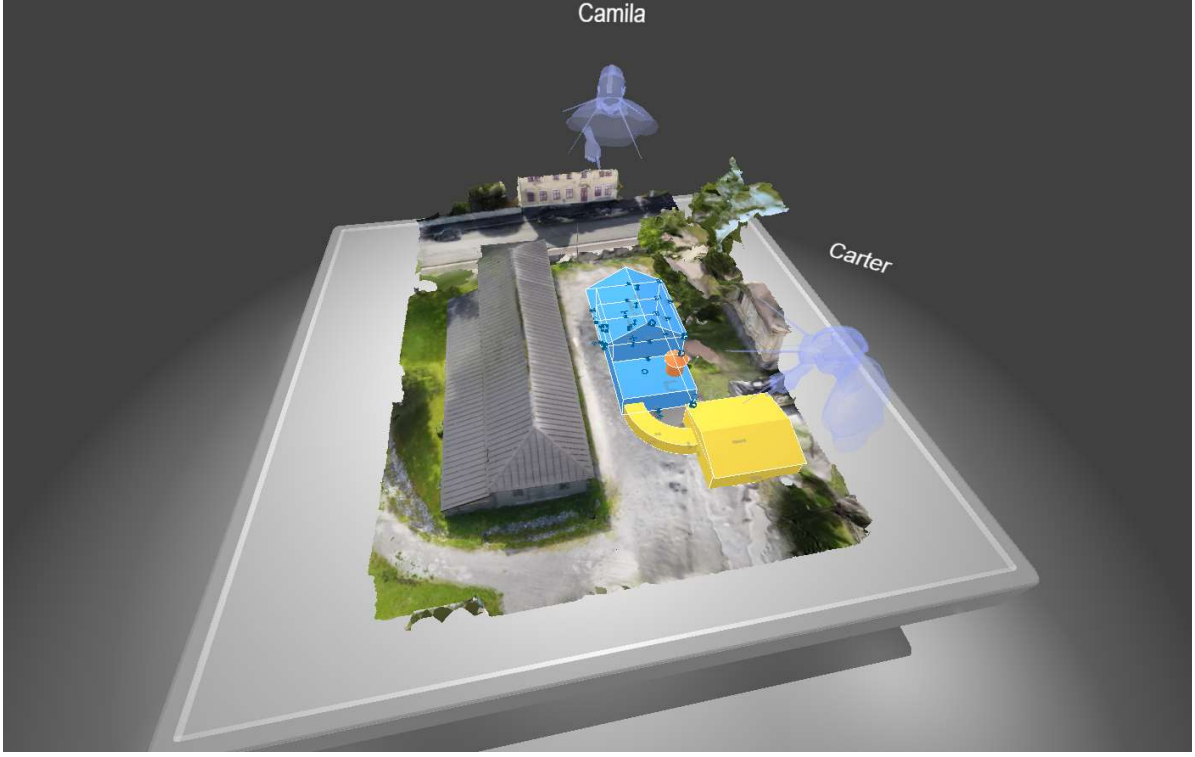


Figure 5: Multiple users collaborate in rooms

3.3 Lobby

The lobby is the first page shown after connecting to the Masser server. Through the lobby the user can join an existing room. With administrative privileges the user can also create rooms.

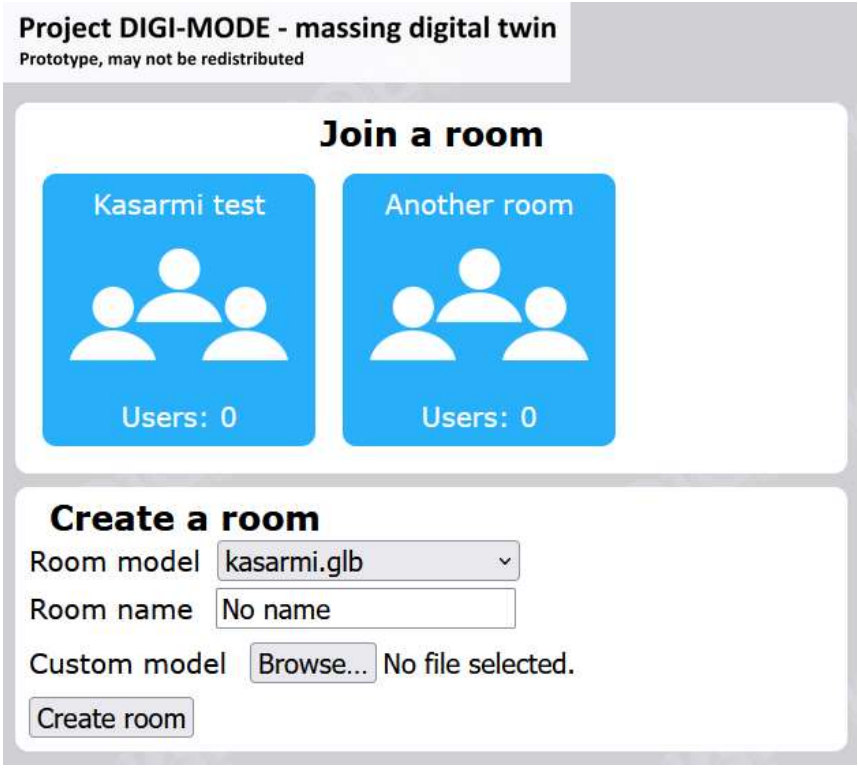


Figure 6: Lobby screen

3.3.1 Joining a room

The lobby lists all the currently available rooms on the server. Rooms can be joined by either clicking the room button or by copy-pasting the room URL from someone already in the room. The URL to a room could be sent via E-Mail or instant messaging to a client, who then joins the room directly without going through the lobby.

Figure 7: Joining an existing room

3.3.2 Creating a room

If the user logged in using administrative privileges, they will be able to create a new room through the "Create a room" panel. A room can be created using one of the existing 3D models on the server (located in `/app/static/models/places/`); or using a custom 3D model uploaded from the local filesystem. See the next section for specific information on how to prepare a custom 3D model for the Masser tool.

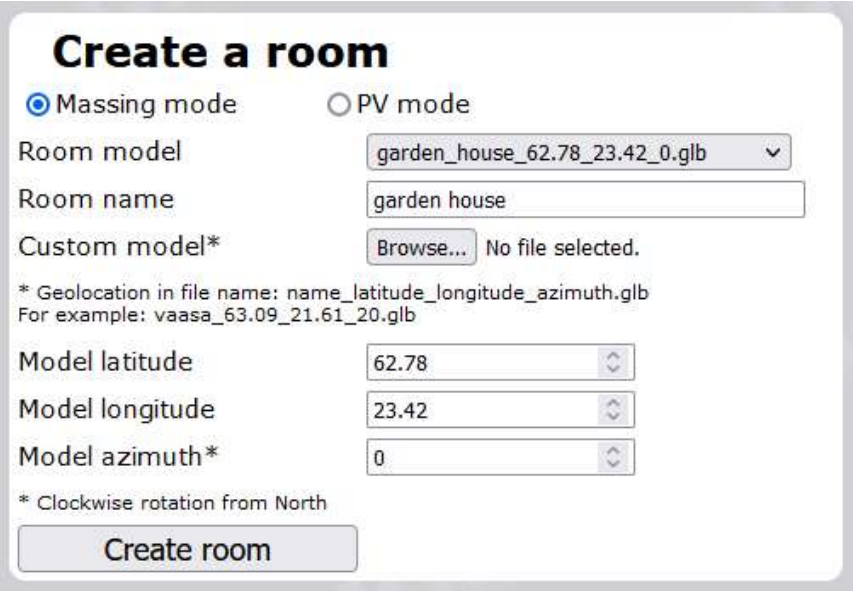


Figure 8: The room creation form

Rooms are automatically closed if they remain empty for 10 minutes.

3.4 Custom model format

Instead of using one of the existing 3D models when creating a room, a custom model can be supplied through the file upload field. The model should be converted to a binary glTF file. The reason why binary glTF files (.glb) are used over plain-text ones (.gltf) is that binary files are able to embed texture data together with the mesh data in a single file.

glTF is a hierarchical format which stores entire "scenes" of objects in the same file. If the model file contains an object named "Environment", it will be treated as a background object which is only visible from a first-person ground view.

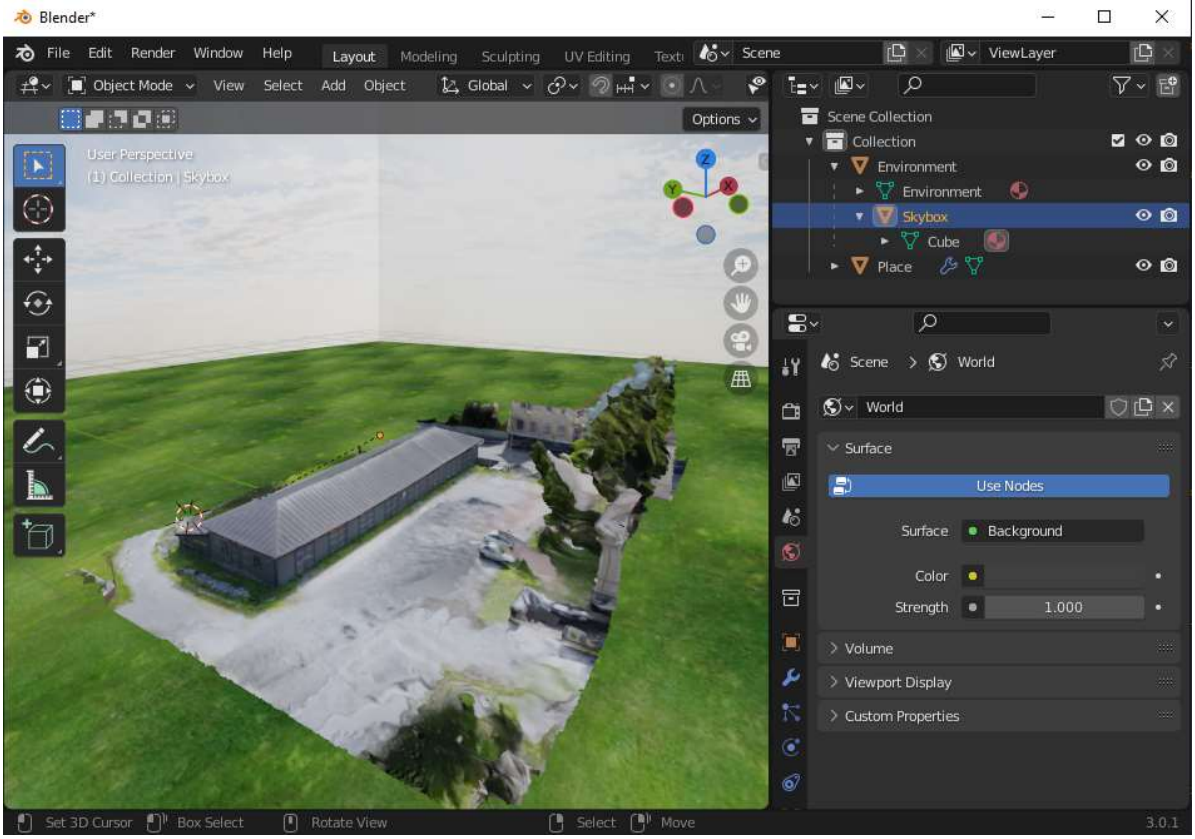


Figure 9: Model with a background object inside Blender

3.5 Inside the room

Once the user has entered the room, they are presented with the room title screen. At this point the user may enter virtual reality by clicking the VR button. Mobile and PC users can start interacting with the room right away.

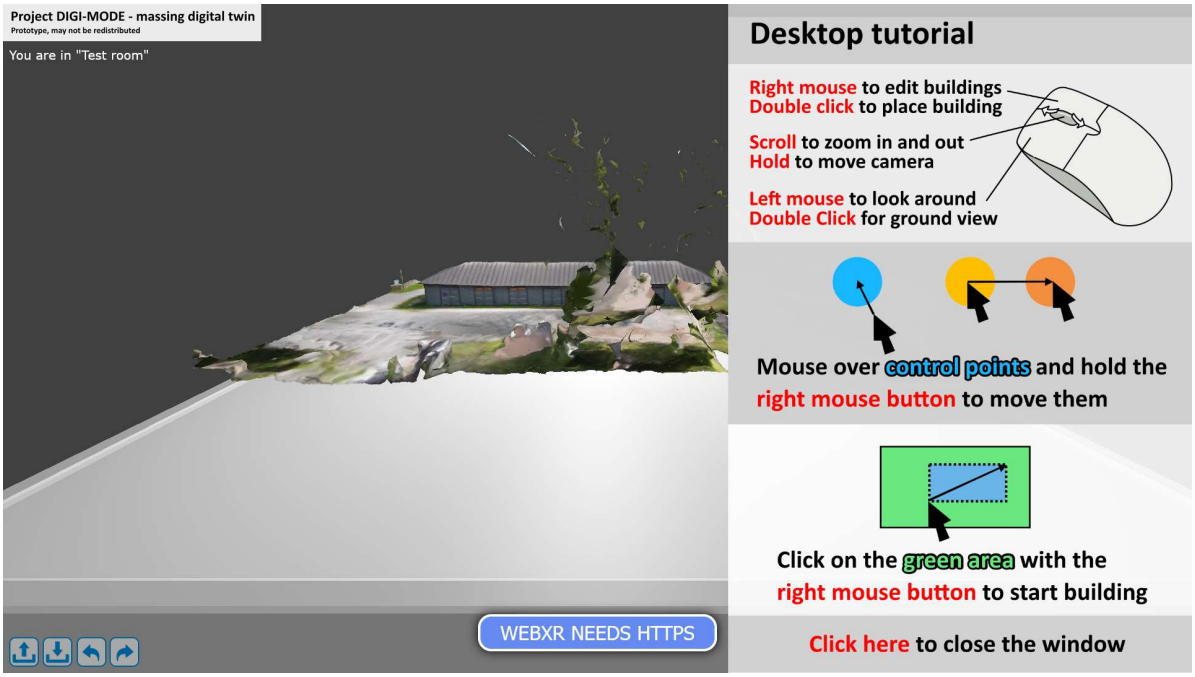
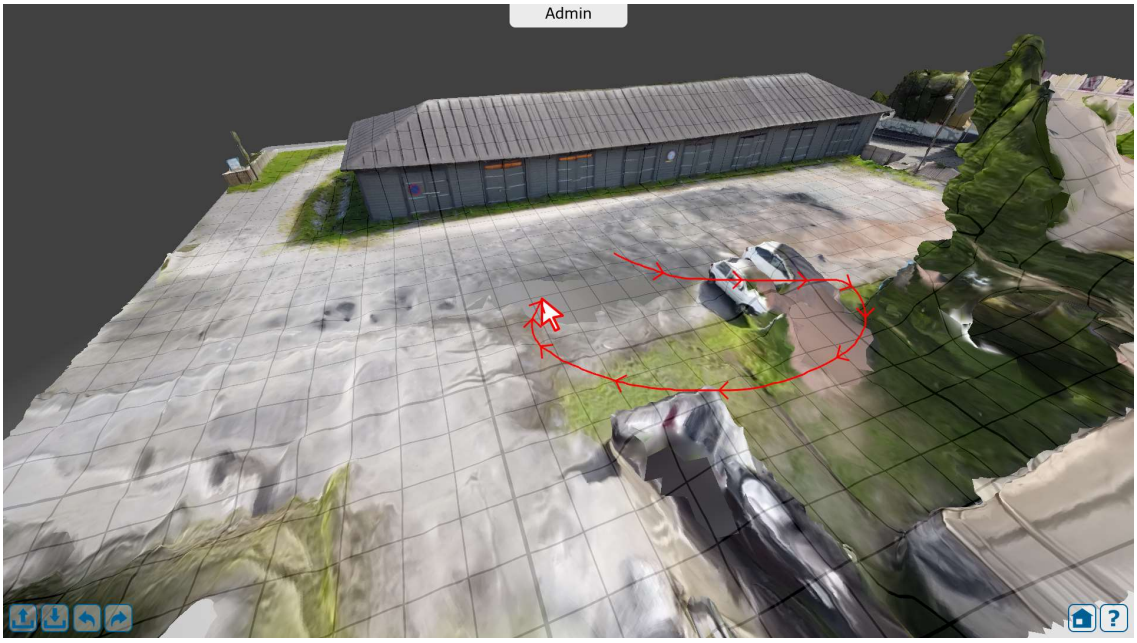
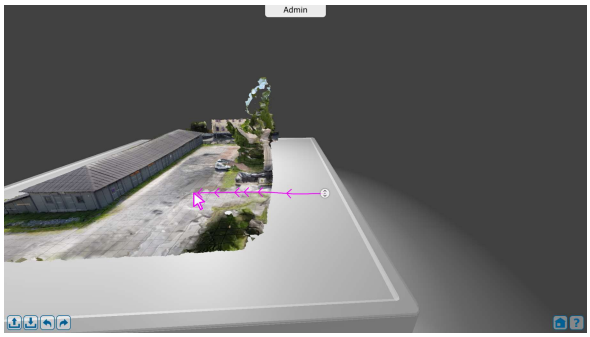


Figure 10: The room title screen

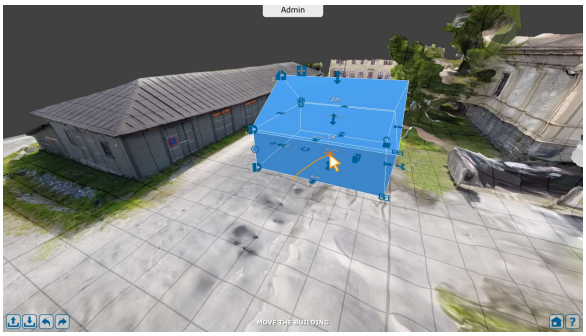
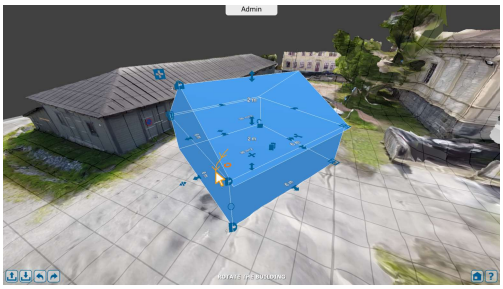
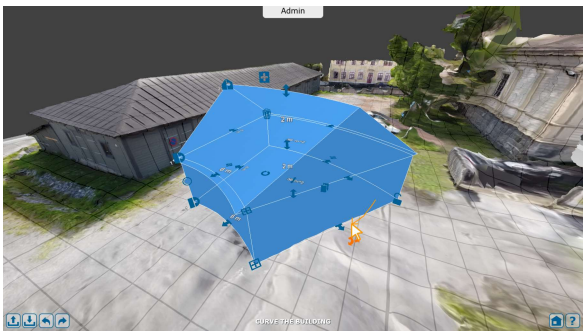
3.5.1 Camera controls

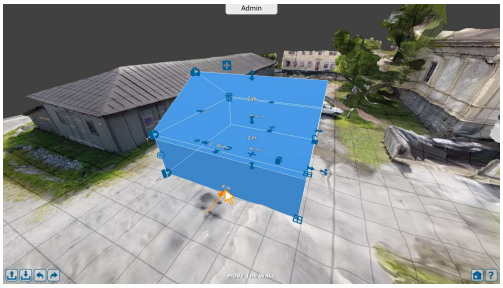
The camera is controlled using the mouse or touchpad.



Hold left mouse to look around	Hold middle mouse to move the camera
	

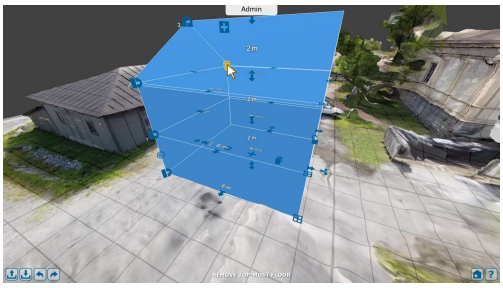
3.5.2 Creating and editing masses

Double right click + drag to create masses	Right click drag to move the mass
	
Right click drag to rotate the mass	**Right click drag to curve the mass along its length**
	
Right click drag to resize side	**Right click drag to resize building**

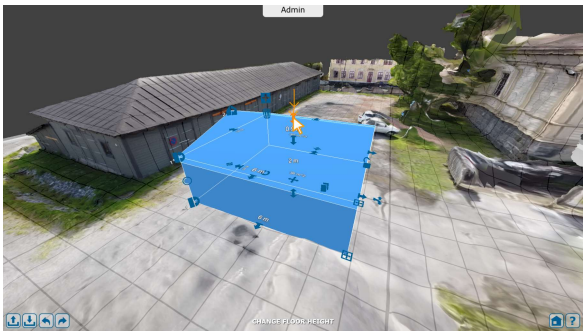


Right click to add a new floor to the mass

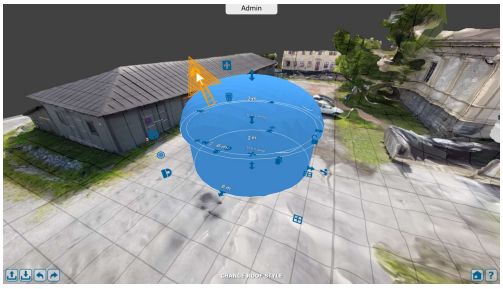
Right click drag to change mass height



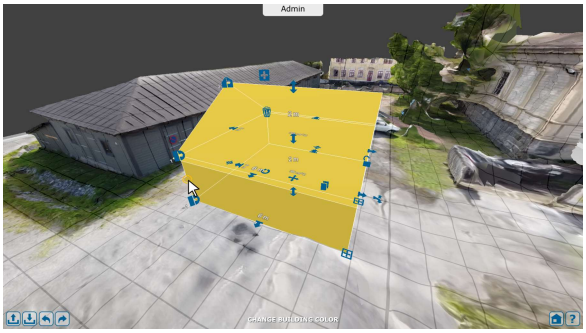
Right click to flip through mass shapes



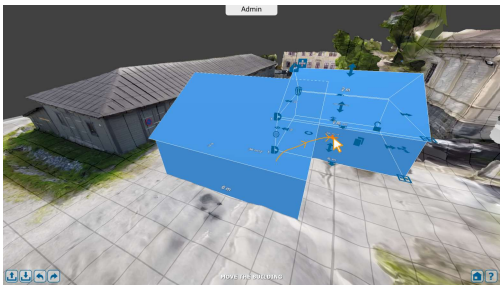
Right click to flip through mass colors

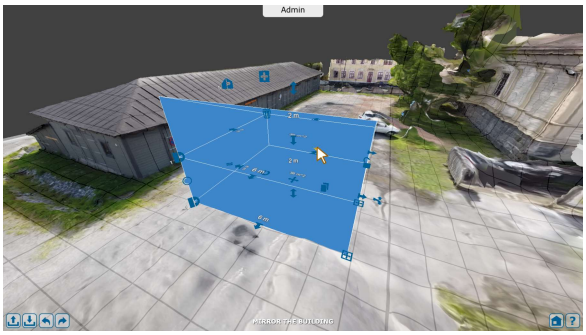


Right click drag to copy mass



Right click to mirror the mass





3.5.3 Cooperation and presenter mode

Multiple users can cooperate in the same room, either as spectators or as active editors.

Users with administrative privileges have access to the admin panel at the top of screen. The admin panel contains a full list of room participants. Build permissions can be set for individual users to only allow certain users to create masses.

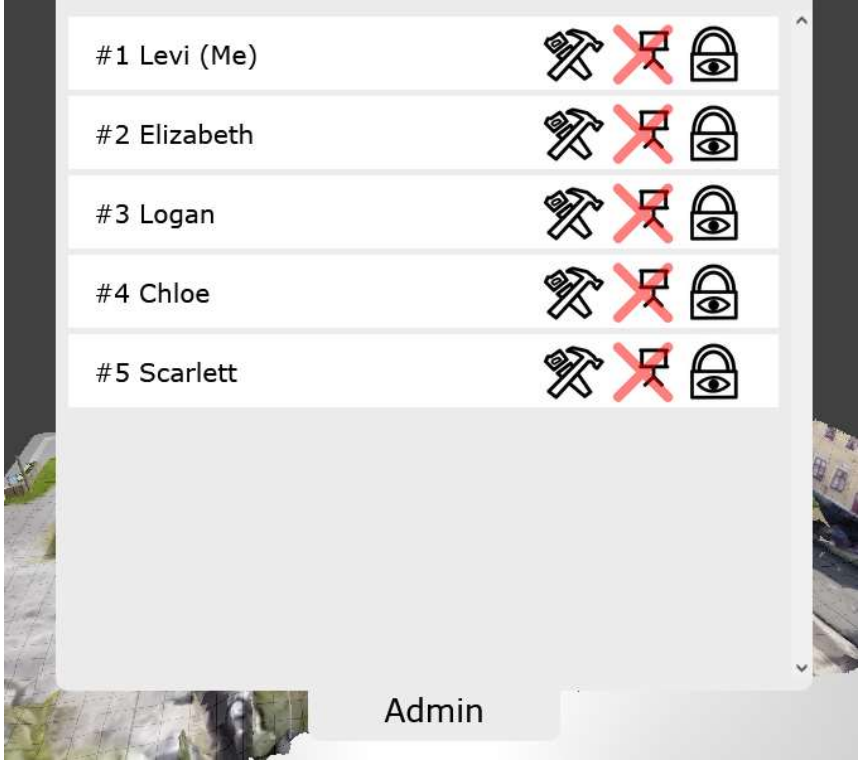
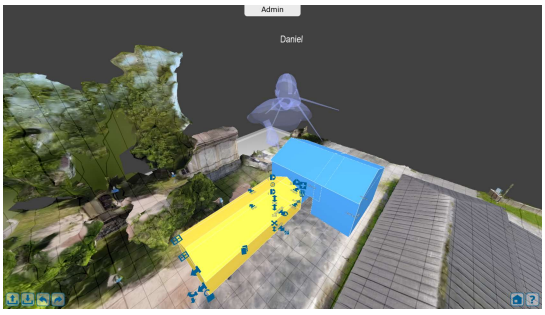
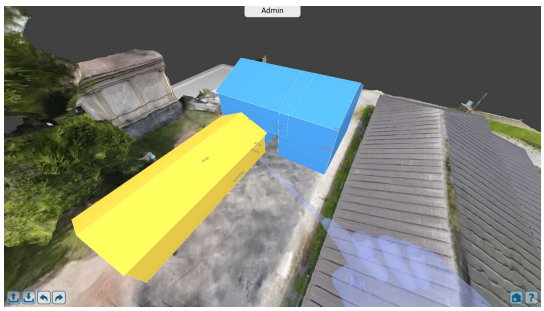


Figure 11: In-app administrative panel

A presenter mode can be enabled by declaring one user as the "presenter" in the admin panel. All other users (except for the ones with view lock disabled and VR users) are forced to watch the room from the perspective of the presenter.

Presenter mode disabled	Presenter mode enabled
	

3.5.4 Virtual reality

Users who join a room using a Virtual Reality compatible headset will be able to enter VR mode using the "Enter VR" button.

In VR-mode, the user can move around using the left joystick, and interact using either of the trigger buttons. The grip button can used to enter and exit ground mode. Everything else works just the same in VR as it does when using a keyboard and mouse.

Figure 12: Oculus Quest

4 Frameworks and web technologies

4.1 HTML5 and Javascript

Applications written to run within a web browser are often called HTML5 applications, to differentiate them from native applications. Native apps are written to use a particular platform such as Microsoft Windows, while a well-developed HTML5 app can be compatible with a large number of modern devices with a web browser, such as computers, smartphones, VR headsets and smart televisions.

HTML5 apps are usually written in the Javascript programming language. Javascript is an old language, originating in 1995. The language has had several updates over the years and has grown to be one of the most widely used programming languages in the world, running most of the modern web.

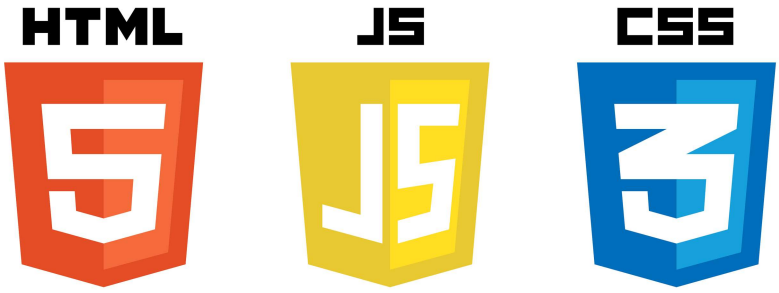


Figure 13: HTML, JS and CSS logos by Clipart.info (CC BY 4.0)

Javascript is an interpreted language with Just-In-Time (JIT) compilation. This means that Javascript code does not need to be pre-compiled into a machine code. The code is compiled from human-readable source code into machine code as it is read inside the web browser. Javascript is not as fast as a compiled language such as C++, but the performance gap has lessened significantly as the Javascript engines used inside the web browsers got faster and more optimized.

WebAssembly is a new binary format which allows Javascript-based applications to run some code at near native speeds.

Javascript and WebAssembly code can also be produced using exporters such as the Unity 3D HTML5 exporter.

4.2 What is an API?

An Application Programming Interface (API) is an interface which allows one program to control another program written by somebody else. The interface contains the definitions and protocols required for communication between programs. In many cases these APIs are public and well-documented, like the different Javascript APIs available in web browsers.

A real-life example of an Application Programming Interface is the Javasscript Web Audio API, through which a website can gain access to the user's microphone and sound system. This is accomplished through the MediaDevices.getUserMedia() function, which is defined in the API specification.

4.3 WebGL

WebGL is a Javascript-based Web API which is used to render high-performance graphics inside a web browser.

WebGL is similar to the APIs used to write Native high-performance applications. WebGL is almost the same as the OpenGL ES2 API. Both native and WebGL-based applications use high-performance shaders to render 3D (or 2D) graphics using the computer's Graphical Processing Unit (GPU, the processor of a graphics card).

WebGL uses the OpenGL Shading Language (GLSL) for shaders. The syntax is almost identical between shader languages and the skillset is easily transferable between WebGL and OpenGL or DirectX development.

WebGL applications can either be written in directly in Javascript, or be exported from a game engine such as Unity 3D, Godot or Game Maker Studio. Exporting the Javascript-application from an extensive engine such as Unity 3D may be detrimental to the overall performance of the application, as these engines are primarily focused on Native builds with a lesser focus on HTML5.

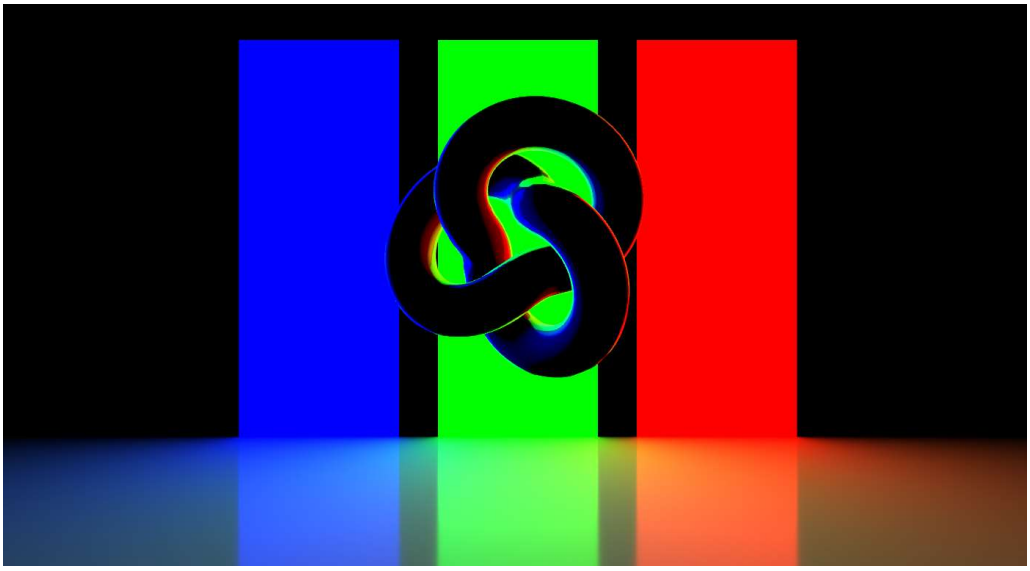


Figure 14: WebGL makes it easy to work with dynamic lights

4.4 WebXR

WebXR is a Javascript Web API for interfacing VR and AR devices with a compatible web browser. Through WebXR, Virtual- and Augmented Reality (Cross reality, XR) content can be served to a compatible device in the form of a website. WebXR is the successor of the WebVR API.

At its core, WebXR handles an XR device input, timing, scheduling, positional tracking and other features required for the Javascript application to interface with the XR device.

WebXR does NOT handle rendering the 3D scene and models. Like with a regular HTML5 3D application, the scene is rendered using the WebGL API and is then displayed inside the VR glasses/headset or other XR device. The WebXR API provides the controller input and headset position, and the Javascript code accounts for the change in the WebGL scene.

Three.js provides a convenient way to handle the 3D scene for a WebXR application.

4.5 Three.js

Three.js is a lightweight general purpose Javascript library for rendering 3D graphics. Three.js uses the WebGL API to render graphics, the Audio API to play spatial sounds and several other browser APIs for other tasks. Three.js also helps with loading 3D models, 3D and matrix math, surface triangulation and many other useful features.

Three.js has grown to be one of the most popular 3D libraries for WebGL, and has been used by many smaller game developers as well as by large agencies such as NASA in their InSight demo. The Three.js website features a large collection of examples and an extensive documentation of the library.

The three-mesh-bvh plugin was used to optimize the native THREE.js raycaster performance.

Figure 15: Three.js

Three.js is compatible with WebXR and works natively on several VR and AR headsets such as Oculus Quest 1 and 2 and Hololens 2.

The A-Frame framework is built on top of Three.js and allows a 3D scene to be defined inside of the HTML markup language without relying on the developer having lot of experience in Javascript.

Figure 16: The House Customizer WebXR demonstration built using Three.js

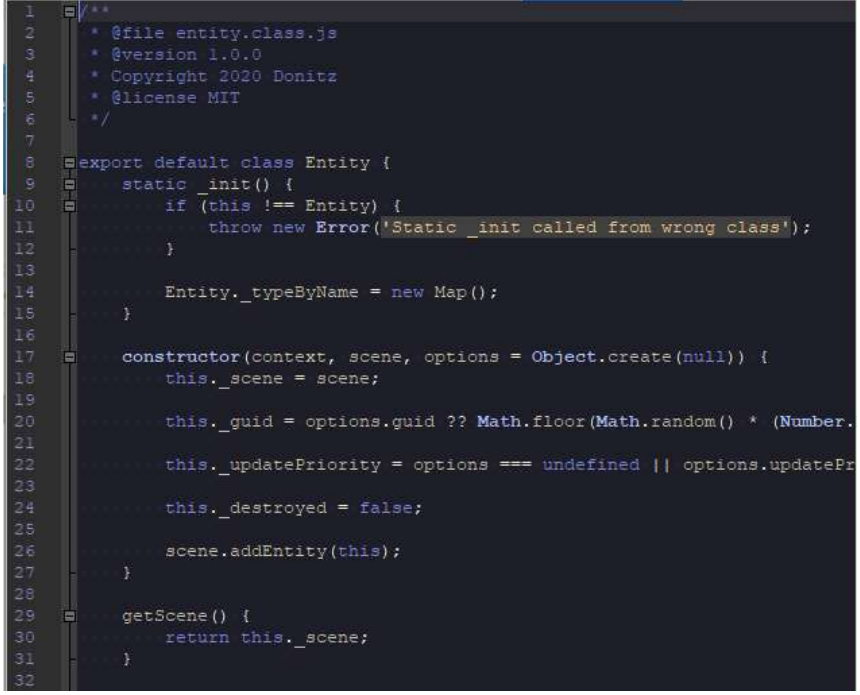
4.6 Node.js

Node.js is an open-source back-end Javascript engine used to run Javascript outside of a web browser. Node.js applications uses the Node package manager to install dependencies. Node.js is often used to host web servers and web services.

Figure 17: Node.js

4.7 Entity system (Dnz Framework)

In addition to Three.js, a minimalistic framework+programming paradigm I (Dennis Bengs) have developed was used to organize the code in the Project DIGI-MODE Masser application. The framework was developed in 2020 and has been licensed under the MIT license. The basic idea behind the framework is that the application is organized into smaller chunks called entities, which are organized into scenes. The scenes provide a convenient wrapper to update, find and organize entities. The framework also provides a simple audio- and input system.



```
/**
 * @file entity.class.js
 * @version 1.0.0
 * Copyright 2020 Donitz
 * @license MIT
 */

export default class Entity {
    static _init() {
        if (this !== Entity) {
            throw new Error('Static _init called from wrong class');
        }

        Entity._typeByName = new Map();
    }

    constructor(context, scene, options = Object.create(null)) {
        this._scene = scene;

        this._guid = options.guid ?? Math.floor(Math.random() * (Number.

        this._updatePriority = options === undefined || options.updatePr

        this._destroyed = false;

        scene.addEntity(this);
    }

    getScene() {
        return this._scene;
    }

```

Figure 18: The core entity class

5 Assets and tools

5.1 3D scanning

One of the initial concepts for the masser app was the usage of drone photogrammetry for scanning and importing real-world places and existing buildings into a digital environment. New digital content could later be added to the model, and parts of the original model could be removed. For example, shrinking a building.

To use a real-world environment inside the Masser application, the area first needs to be scanned and converted into a digital 3D model. There are multiple methods to capture a 3D environment, including but not limited to:

- Photogrammetry using 2D photography
- LiDAR scanning using lasers (Point-cloud data)
- Structured-light scanning using special 3D scanners

Photogrammetry is a method of generating 3D models and taking measurements using regular consumer cameras, or drone-mounted cameras. Many photos are taken around the area or subject from different angles. These photos are then processed by sophisticated computer software which turn them into a 3D model. The tool used to create the 3D environments for the Masser tool was Agisoft Metashape, a commercial photogrammetry suite.

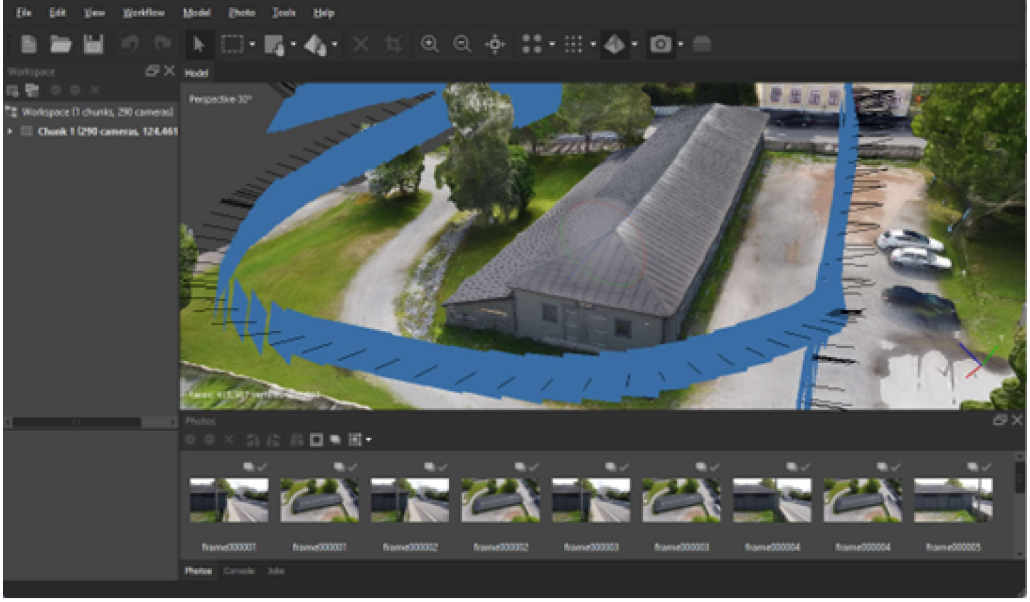


Figure 19: 3D reconstrution of the old Vaasa barracks using Metahape

Photogrammetry works by cross-referencing multiple photos to find multiple matching points in two or more photos. These points could be features in the photo such as corners of a building. The 3D position of these points can be estimated using triangulation between multiple photos. With enough matching points it is possible to reconstruct a 3D model, which the original photo colors can be mapped onto.

The example area shown in this documentation is created using photogrammetry. A drone was flown over and around the Kasarmi building while capturing a large number of photographs. These photos were later processed by a photogrammetry software to produce the 3D model seen below.

Figure 20: The Kasarmi 3D model

5.2 MakeHuman

The player model used in the masser app was procedurally generated using an open-source tool called MakeHuman. With MakeHuman it is possible to customize and export 3D meshes of procedurally generated humans. This is a convenient way to quickly generate 3D humans for tools, demonstrations, games and animations.

The exported human was post-processed in Blender to create the final model for the masser.

Figure 21: Creating the player model using MakeHuman

5.3 3D editor

Additional 3D assets were created using Blender. Blender is a free and open-source 3D editor. It is cross-platform and can be used in Microsoft Windows, macOS and Linux. Blender is a powerful and versatile tool, and can be used to create 3D models, 3D environments, animations, fluid- and cloth simulations and photorealistic art. Blender can even be used as a movie editor.

Blender is also one of the harder tools to learn to use efficiently. The editor relies heavily on keyboard-oriented modelling and shortcut memorization. If you want to learn Blender, this tutorial is a good place to start.

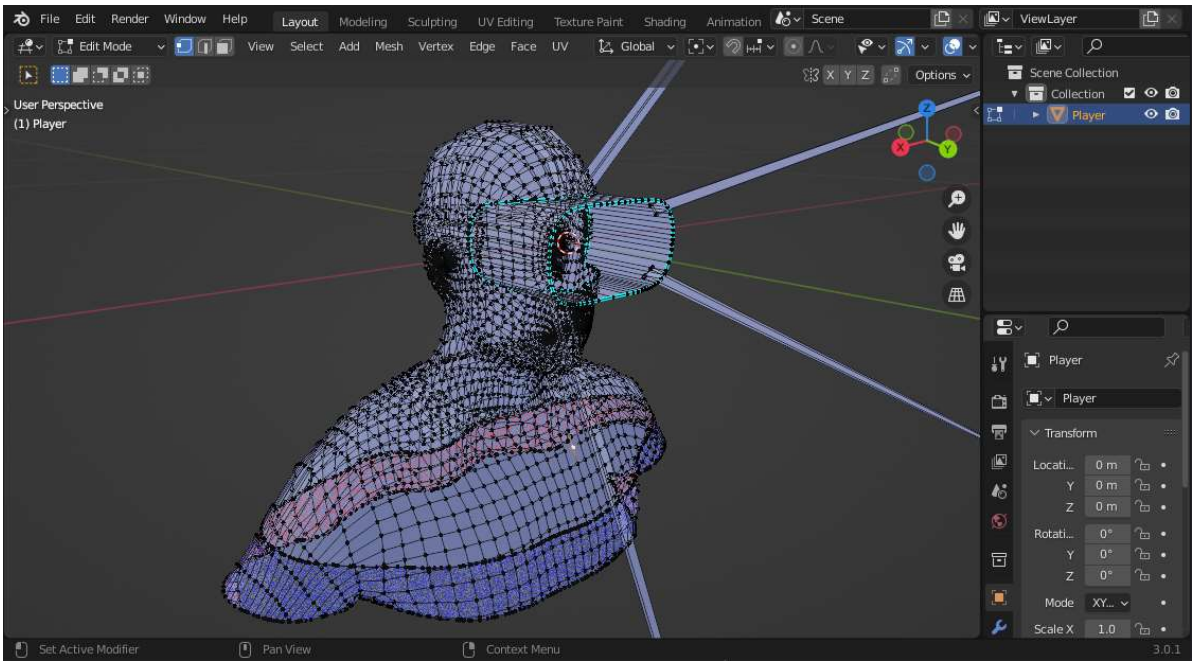
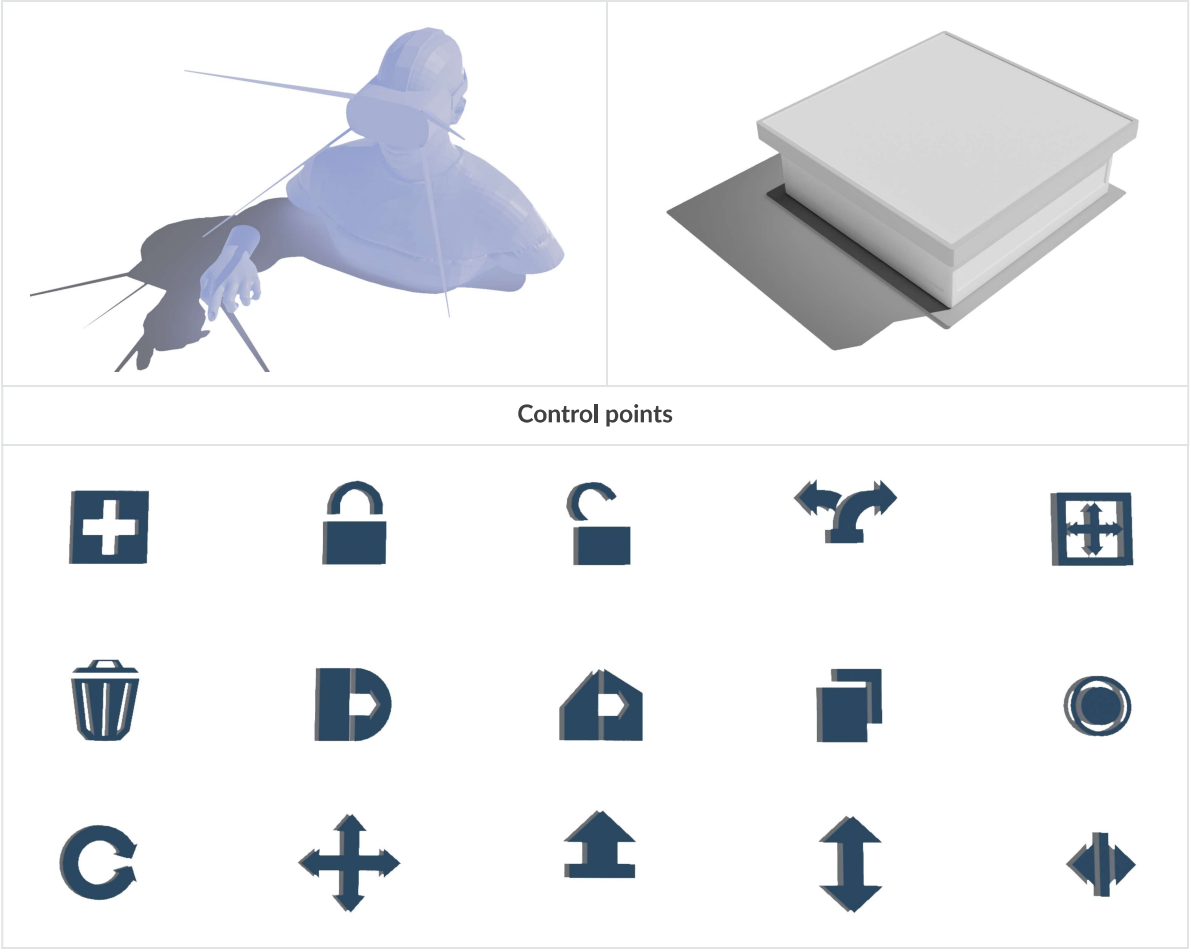


Figure 22: The player model inside Blender

Multiple 3D models were created from scratch for the Masser application, such as the table and the graphical icons shown when editing masses.

Player model	Table model



Blender was chosen to design icons due to the need to draw them inside a 3D environment and at a close-up to the player camera. Using 3D meshes instead of images allows for perfect rendering of the icons at any distance from the camera.

A number of different masses were designed in Blender to act as a starting point for the transformed masses shown inside the app. These 3D modelled masses contain an abnormally large number of extra vertices. These vertices are needed when later morphing the masses inside the tool. Different masses were tested together with NAC Arkkitehdit to pinpoint the fewest and most useful masses would look like, and eventually we came up with the following four masses:

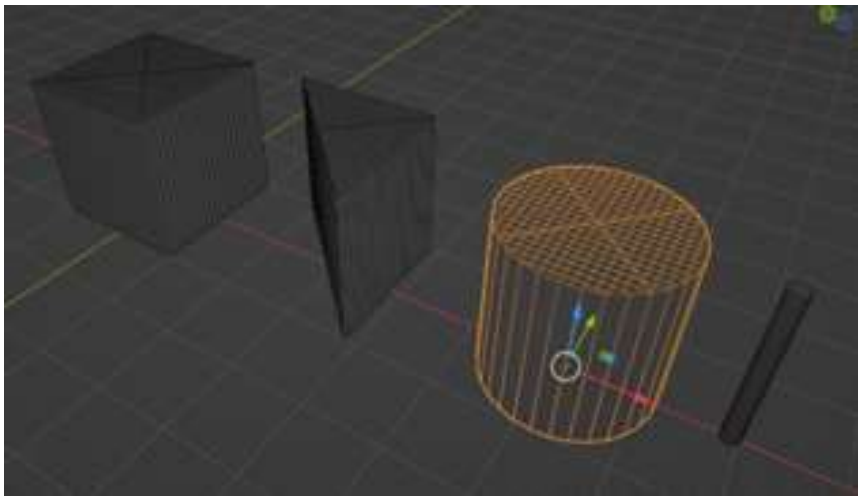


Figure 23: Base masses from which composite masses are created

5.4 Sound repositories and editor

The popular sound repository freesound.org was used as a source for sound effects for the Masser. Freesound contains music and sound effects licensed under the different Creative Commons licenses.

Further audio editing was done in Audiocity, an open-source digital audio editor.

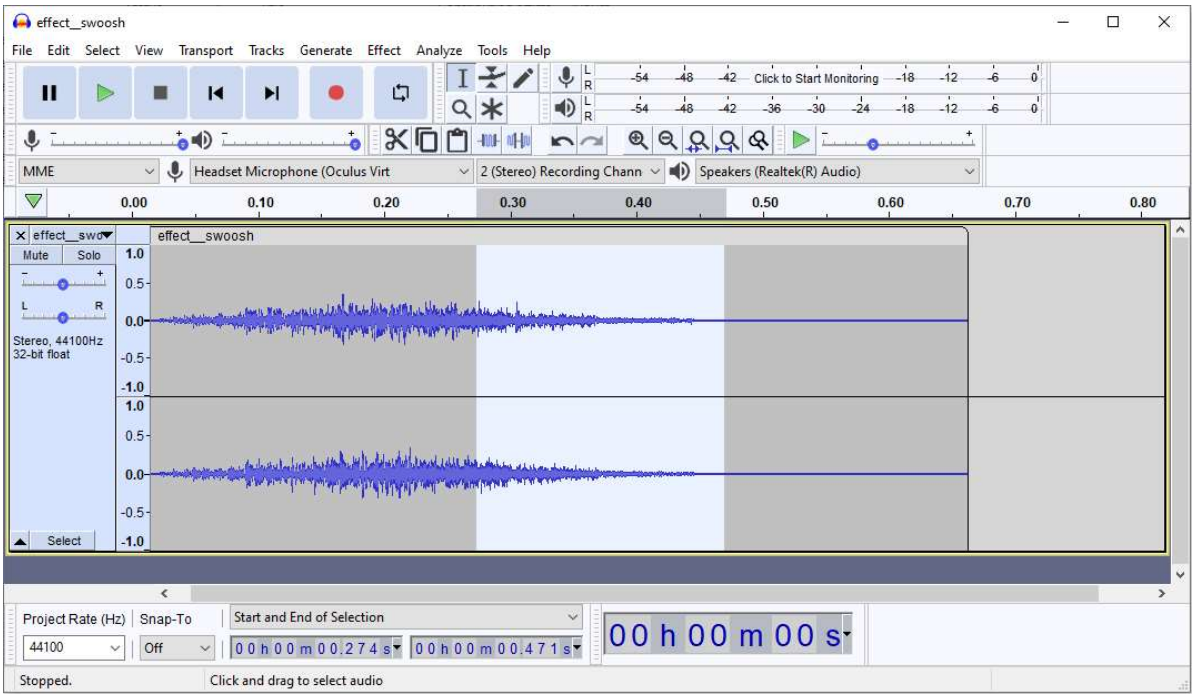


Figure 24: Audacity lets you fine-tune your effects

5.5 Source-code editor

All programming was done in the open-source text editor Geany. Geany is lightweight, cross-platform and supports syntax highlighting for over 50 different programming- and scripting languages.

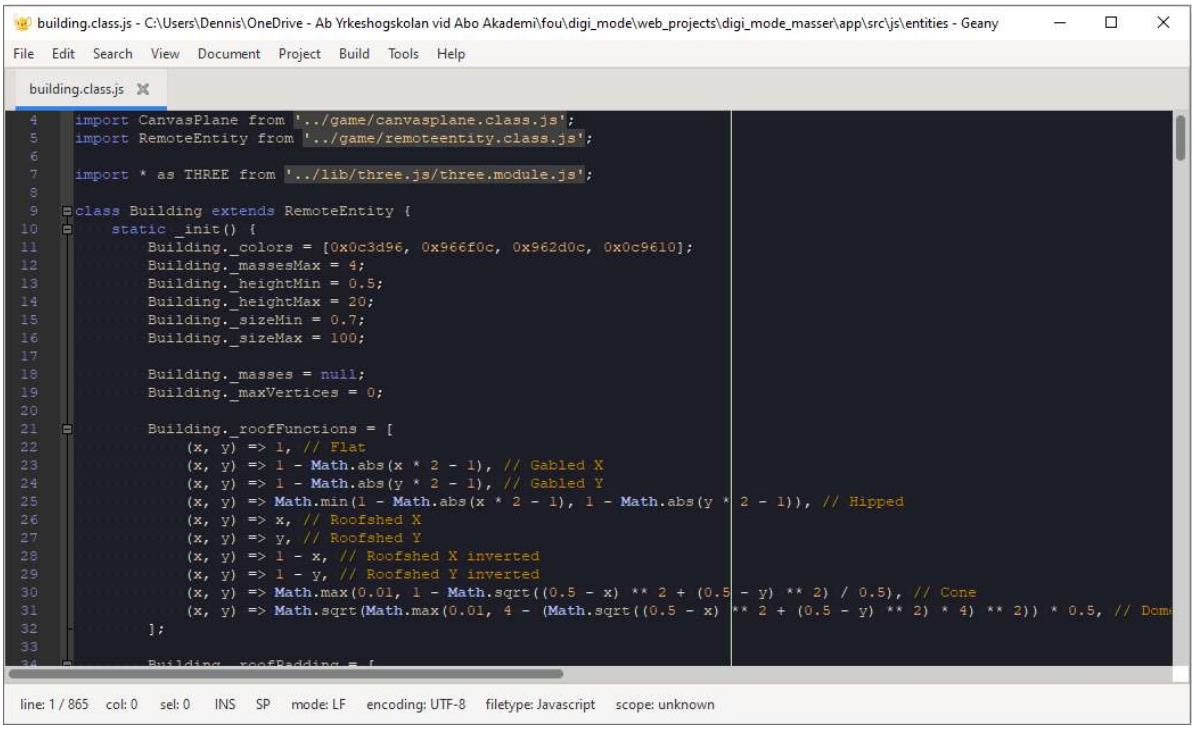


Figure 25: A Javascript file in Geany

6 Development

The Project DIGI-MODE Masser tool was developed in cooperation with NAC Arkkitehdit. The tool was developed in several sprints (A sprint is a short period of time in which a team works to complete a set amount of work). At the end of each sprint, we had a meeting with NAC Arkkitehdit and got feedback on the current progress of the tool. New changes and features are proposed during this meeting and are implemented in the next sprint.

6.1 Deciding on a game engine/3D framework

The first step in developing the tool was to select a game engine/3D framework. Several engines were considered, including:

- Unity 3D
  - **Pro**: Popular with a large community
  - **Con**: Bloated editor
  - **Con**: Confusing licensing for academic projects
- Unreal Engine
  - **Pro**: Free for non-commercial use
  - **Con**: Bloated editor
  - **Con**: Difficult programming language
  - **Con**: Badly documented
- Godot
  - **Pro**: Open-source
  - **Pro**: Simple node system
  - **Pro**: Lightweight editor
  - **Con**: Relatively slow
- THREE.js
  - **Pro**: Open-source
  - **Pro**: Lightweight
  - **Pro**: Based on WebGL (runs inside browser)
  - **Pro**: Ease to share (No install required)
  - **Pro**: Supports VR
  - **Con**: No visual editor, only code

We decided to use THREE.js due to the ease of portability and sharing when developing a collaborative tool. We also have experience from using THREE.js in earlier VR-based projects with a lot of success.

6.2 Project structure

The Masser tool is split into two components:

- The server which hosts the web application, facilities communication between clients and keeps track of the rooms.
- The web application which renders the rooms in a web browser.

6.3 Agile software development

In agile software development the work is broken up into short sprints in which the developers are committed to a certain task. At the end of each sprint a retrospective meeting is held to review the work done.

The masser app was developed over the course of months in short two-week sprints. At the end of each sprint, we scheduled a meeting with NAC Arkkitehdit. During the meeting we demonstrated and reviewed the last sprint's progress and planned improvements and new features to be developed during the next sprint.

The majority of the masser app was developed in the first few sprints, and after that changes become more gradual improvements, adjustments and new features. At the later stages of the development process the project team did usability testing at Experience Lab.

6.4 The first sprint

The initial version of the tool was developed in the first sprint.

In this version masses could only have a single floor each. The floor and ceiling height could be changed independently. The placement and size of the masses were based on in their corners which got a bit confusing.

The networking component was written in the first sprint. This is the part of the code which allows multiple users to see each other and interact with the world over the Internet.

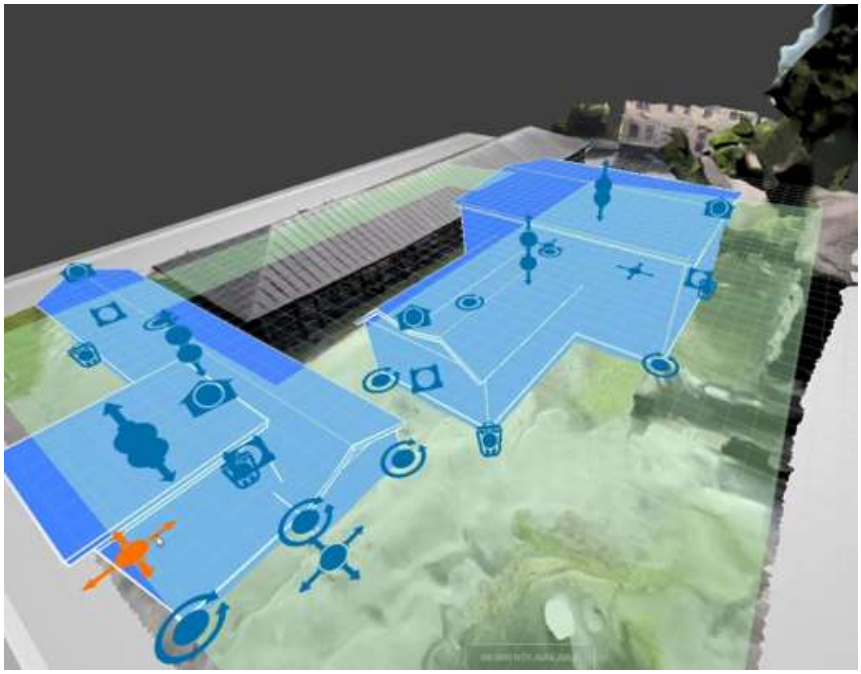


Figure 26: Sprint 1

6.5 The second sprint

In the second sprint more types of masses were added, and the user interface was improved with more options for editing masses.

6.6 The third sprint

The user interface was re-written and improved in the third sprint, with more sophisticated control over the placement of buildings. Many new types of masses were added. The overall code of the application was also refactored at this point to be cleaner and easier to work with.

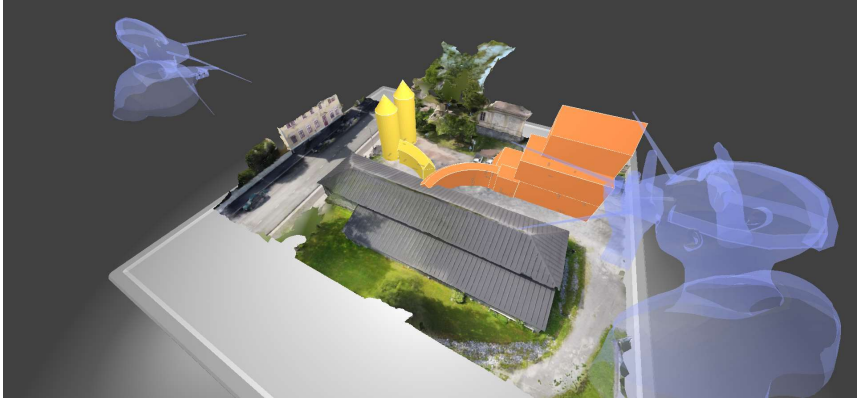


Figure 27: Sprint 2

6.7 The fourth sprint

In the fourth and final major sprint even more advanced controls were added to masses, such as the ability to curve and copy masses. A markdown documentation was written for the masser.

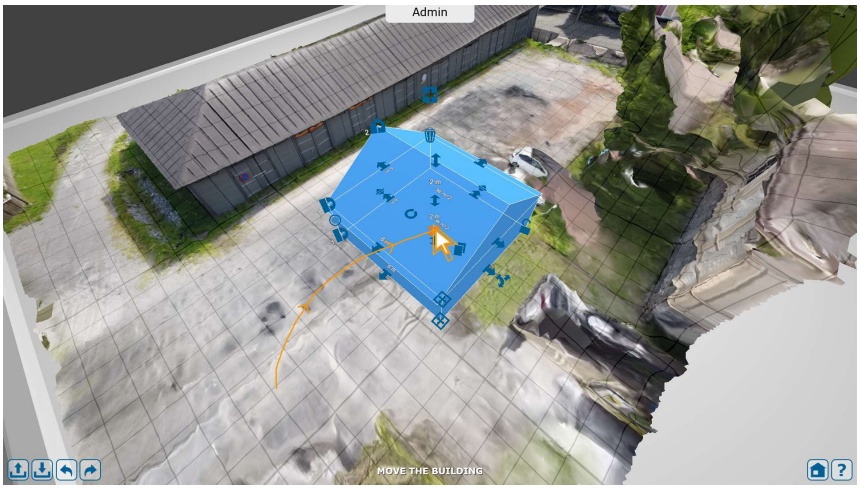


Figure 28: Sprint 4

6.8 Retrospective meetings and UX testing

A retrospective meeting was held at the end of each sprint. During these meetings we demonstrated our progress to NAC Arkkitehdit and to the project group as a whole. Any feedback over improvements, issues and other notes about the current version of the masser were documented and a to-do list was created for the next sprint.

User experience testing was done at the later stages of development, which involved unsupervised testing of the current version of the masser.