DELFT3D

Development Report

Synthesis Project 2024

Adhisye Rahmawati	-	5890837
Ákos Sárkány	-	5086086
Hyeji Joh	-	4995945
Lars Boertjes	-	4704541
Rafal Tarczynski	-	4439899
Xueheng Li	-	6001114

8. November 2024

TU Delft | Gemeente Delft

Abstract

This report documents the development of Delft3D, a web-based 3D application, designed to visualise Delft's urban features—such as buildings, vegetation, and water bodies—using open datasets. Developed for the Municipality of Delft, Delft3D customises the Netherlands3D platform to provide accessible, interactive visualisation for residents and employees of the municipality. This report primarily focuses on the process of development and only explains the implemented features conceptually. After the explanation, we provide a concise reflection on the hurdles we faced during the project and we give an overview of the limiting factors.

KEYWORDS: Visualisation, Digital-Twin, Unity-WebGL, Netherlands3D

Contents

Figure 1: Viewport of the Delft3D application.

1 Introduction

Urban visualisation nowadays is an up-and-coming topic in city planing. With the technological resources of the 21^{st}. century, it is now possible to generate 3D models for entire countries given the necessary data. The Netherlands3D project was initialized to develop a 3D Digital Twin for the Netherlands, which then can be adopted by municipalities.

In this project, we aim to develop an online 3D viewer application for the Municipality of Delft: *Delft3D*. This is done by reusing, configuring, and complementing the existing Netherlands3D framework. The actual functionalities we implement for this project are based on our discussion with our client.

The primary stakeholders in the project are the Municipality of Delft and the Delft University of Technology (TU Delft). The client of this project is the Municipality, where their requests determine the project requirements. At the same time, this project is an integrated part of the Synthesis project course at TU Delft. Thus, the priority of the final requirements are also influenced by the goals of the course. Further stakeholder is the Netherlands3D project itself, in which we directly test their application.

In the next section, we present the methodology we followed throughout this project. In Section 3, we discuss all features we added to or configured to derive the Delft3D application. Section 4 reflects on the results we achieved in this project and compares them to the original requirements. Finally, in Section 6 we elaborate on our observations regarding this project.

2 Methodology

This section outlines the methodology we used throughout the development of Delft3D. This includes our project management framework, stakeholder involvement, requirements derivation, prioritisation, and a review of relevant projects.

2.1 Project Approach

To manage the project's development, we followed the SCRUM collaboration framework to organize the project into iterative sprints and continually adapt to feedback from stakeholders.

SCRUM Model Implementation

Our implementation of the SCRUM model involved several steps:

- **Requirements Derivation**: In the first two weeks, we discussed the desired functionalities with our client, the Municipality of Delft, and reviewed the outcomes in meetings with our TU Delft supervisors.
- **Schedule Creation**: Based on the derived requirements, we used the MoSCoW prioritization method to define the project scope and created a project schedule with specific deadlines for each component.(as shown in Figure 6 in Appendix D).
- **Weekly Updates**: The project was organized into weekly sprints, where tasks were distributed to subgroups and monitored. In weekly meetings, we reviewed the previous sprint's progress, identified challenges, and adjusted tasks for the next sprint.

2.2 Requirement Engineering

To prioritise the importance of each requirement, we employed the MoSCoW method, categorising them according to their respective levels of necessity. This categorisation is presented in Table 1, where each requirement is assigned a level of priority. By applying the MoSCoW method, we structured the requirements as follows, with a thorough explanation of each specific requirement available in Appendix A.

Category	**Requirement**
Must	Terrain (1) Buildings (2) Vegetation (3) Ground Surfaces (4) 2D Layers (5)
Should	Model Upload (10) 3D Masterplans (6) 2D Masterplans (7) Data Synchronisation (12)
Could	Temporal Slider (9) Water Levels (11) Underground Utilities (8)
Won't	Flood Models (13)

Table 1: Requirements using MoSCoW method.

2.3 Review and Analysis of Related Platforms

To guide our design and development, we reviewed three related 3D platforms: 3DAmsterdam, 3DUtrecht, and Netherlands3D. The following outlines the key characteristics of each platform.

- **3DAmsterdam**: 3DAmsterdam provides a 3D view of Amsterdam, designed to improve urban engagement and visualization. It offers tools for exploring city information and visualizing spatial changes, with a focus on accessibility through WebGL. Developed initially as an independent project, development for 3DAmsterdam has now been consolidated under the Netherlands3D repository. For further information, the 3DAmsterdam project repository is available at 3DAmsterdam GitHub.
- **3DUtrecht**: 3DUtrecht is a web-based platform that allows users to interactively explore Utrecht in 3D using WebGL technology. Forked from the 3DAmsterdam codebase, it shares much of its foundational structure—such as core rendering, interaction capabilities, and data handling. However, 3DUtrecht's code has also been consolidated into the Netherlands3D repository to streamline updates and feature development. For additional details, see the 3DUtrecht project repository at 3DUtrecht GitHub.

- **Netherlands3D**: Netherlands3D is an open-source digital twin framework developed to provide a unified 3D environment for visualizing datasets across the Netherlands. It allows users to select specific data sources and functionalities based on project needs, supporting formats such as GeoJSON and WFS. Standardized datasets, including 3D BAG and BGT and Het Kadaster data, create a consistent 3D model of the Netherlands. Netherlands3D also supports custom environments with specific functionalities and starting locations, and it is designed to facilitate collaboration with external developers. More information is available on the Netherlands3D GitHub.

Based on this review, we decided to fork the Netherlands3D repository as the foundation for Delft3D. Our main reason is that from the Netherlands3D developers, we understand that while the project is still in its early stages and lacks certain functionalities and documentation, the 3D Netherlands platform will continue to evolve and be updated. Looking to the future, we opted to choose Netherlands3D as our framework to build Delft3D upon.

3 Implementation

In this chapter, we will outline the steps taken to develop Delft3D, including environment setup, project initialization, configuration, and the added features and customizations specific to the project.

3.1 Setting Up the Development Environment

Unity was chosen as the development environment for Delft3D because the Netherlands3D framework, which serves as the foundation for Delft3D, was developed in Unity. Since Unity uses C# for scripting, understanding the basics of Unity and C# became essential for customising and extending the Netherlands3D framework to meet Delft's specific requirements.

As Unity and C# were not covered in our coursework, we utilised a combination of official resources and community tutorials to develop a foundational understanding. The following resources were especially helpful in setting up and learning Unity:

- **Unity Learn**: Unity's official learning platform, providing tutorials from beginner to advanced levels. The "Create with Code" course introduced C# scripting basics, object-oriented programming principles, and Unity's core components.
- **Unity Documentation**: The official documentation helped in understanding Unity's classes, methods, and components, especially when working with Unity's Universal Render Pipeline (URP) and WebGL for optimising the 3D environment.

3.2 Project Initialisation

Creating a New Project from Netherlands3D

The first step involved initialising a new Unity project by forking the Netherlands3D repository. This process provided the structure for Delft3D, allowing us to build upon the existing codebase while making modifications specific to Delft's data and visualisation requirements.

To understand the setup steps for initialising the project, we referred to the Netherlands3D documentation, specifically the section on Project Loading and Replacement. As the documentation is an ongoing project, additional efforts were necessary, such as exploring Unity independently and arranging meetings with the Netherlands3D development team to clarify specific aspects of the framework and integration process.

Repository Fork and Image Customization

After forking the Netherlands3D repository, specific modifications were made to customize the platform for Delft3D.

3.3 Configuration

2D Layers

As outlined in the detailed requirements section in Appendix A, the 2D layers functionality should enable users to project both national and local datasets over the 3D model. These layers can be either directly integrated into the application or accessed through WMS (Web Map Service) or WFS (Web Feature Service) services.

The Municipality of Delft provided two lists: an initial catalogue of WMS and WFS services for integration into Delft3D (see, Appendix B), followed a few weeks later by a list specifying which WFS layers should be excluded from the final viewer (see, Appendix C). This original list was trimmed down after we presented the viewer, containing all the WFS layers, in an intermediary meeting with them. During this meeting, they reconsidered which layers they actually wanted, now that they had seen them in the viewer. For example, they did not consider "Main and secondary salt spreading routes" important for the public or the users of the viewer. Additionally, layers such as "Wateren" and "Grassen" were already implemented in the viewer through the base terrain layer. All WFS services referenced are accessible through the Delft Open Data Portal, a municipal platform providing free access to the city's open datasets.

Currently, the Netherlands3D framework supports only WFS layers in GeoJSON format; WMS layer import functionality is not yet available. However, the Netherlands3D team plans to implement this feature in the first quarter of 2025 (Q1 2025), with a workflow similar to the existing WFS layer import process.

Since the WMS functionality was not yet implemented, we attempted to create a workaround ourselves. Initially, we explored the UrbanRelief functionality, which uses images projected with a DecalProjector. Another option was the `MapOverlayLoader.cs` script in the indicators legacy package, which loads a texture remotely and can link to a DecalProjector through events. Unfortunately, both options only worked with remote PNG images and could not be connected to a WMS link. After investing considerable time in these workarounds, we decided to discontinue them. The primary reasons were that the municipality required this feature to display their 2D urban masterplans, which by week 8 were still unavailable. Additionally, this workaround would only be temporary, as the official WMS functionality is currently under development and expected to be operational in a few months.

In Unity, the Netherlands3D framework displays 2D WFS layers by parsing GeoJSON data and visualizing various geometries—such as points, lines, and polygons—over a 3D terrain. The framework uses custom shaders to project these 2D features accurately onto the terrain surface, leveraging Unity's UV coordinates and color mapping to align data with the underlying 3D geometry. Each feature is processed and rendered according to its coordinate reference system.

Each loaded GeoJSON layer is instantiated into a 'GeoJsonLayerGameObject', which then creates specific game objects based on the relevant data type:

- `GeoJsonLayerGameObject`

  A Unity class that reads GeoJSON files containing spatial data (points, lines, polygons) and generates corresponding game objects in the Unity scene. It acts as a parent class for each geometry type.

- `GeoJSONPointLayer`

  It handles point features from GeoJSON files, placing Unity objects at specified geographic coordinates.

- `GeoJSONLineLayer`

  It reads line geometries from a GeoJSON file and renders them as GameObjects in Unity using line rendering techniques. It connects points to form lines, representing features like roads or borders.

- `GeoJSONPolygonLayer`

  It renders polygon features from GeoJSON files, creating meshes for areas like building footprints or land parcels. Using Unity's mesh renderer, it generates polygons from the GeoJSON coordinates.

- Visualization `Gameobject`

Finally, a visualization script takes all the attributes belonging to a specified game object of the types mentioned above and visualizes them in a new visualization game object. For example, for a polygon, it creates a PolygonVisualisation game object.

Netherlands3D offers two ways of importing WFS layers, through their URL or by uploading a local WFS file. The process is explained in detail in the Delft3D's documentation 2024.

The WFS functionality of Netherlands3D is still in its early phase, lacking certain symbology and customization options. As of this writing, it is only possible to load WFS layers, such as polygon layers, which are displayed in a single color over the 3D geometry. Even when a layer consists of multiple polygons, like the 'Buurten' layer, it is rendered as one large polygon and does not distinguish between the individual sub-polygons in terms of color (see Figure 2). Additionally, in the layers legend, you can only toggle the layer on and off; changing its color is not yet possible with the current Netherlands3D framework. We have made some improvements to enhance this visualization for the user:

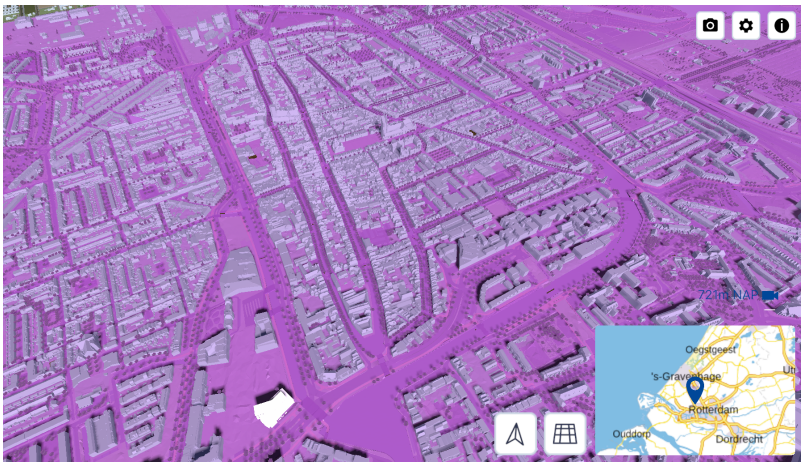


Figure 2: 'Buurten' WFS layer, no colour distinction between polygons

Figure 3: 'Buurten' WFS layer, randomized colour polygons

For layers with disjoint polygons, such as the monuments layer (see Figure 4), we retain the same color, as the disjointed nature of the polygons already distinguishes certain features from each other.

These changes all took place within the `GeoJSONLayerGameObject.cs` script, which builds the visualization of the polygons, ultimately creating a visualization game object. For our final product, these changes were beneficial, as the ease of viewing layers like "buurten" has significantly improved (see Figures 2 and 3). However, in conversations with the developers of Netherlands3D, they emphasized that in the future, symbology and additional customization options will be added to the WFS layers within the `GeoJSONLayerGameObject.cs`, which will likely involve incom-

Figure 4: Monumenten WFS Layer

patible changes. Therefore, this solution serves as a temporary bridge while the function still lacks customization and symbology options.

To streamline the management of Delft3D content, we implemented a hierarchical folder structure for WFS layers sourced from the Open Data Portal. This organizational improvement allows users to control visibility at both individual and group levels, enhancing the overall usability of the viewer.

One aspect the user will notice when starting up the Delft3D viewer is that all layers are toggled on by default. The `LayerToggle.cs` script is designed to control layer visibility. Within it, the `OnEnable()` function checks whether the toggle is on and whether the `layerGameObject` is present. If the toggle is off and the `layerGameObject` exists, it should destroy the `layerGameObject`. In other words, if a layer is toggled off, the corresponding `GameObject` should be automatically removed.

The code we implemented to address this behaviour is, according to the developers, correct. However, they noted that a known bug persists, requiring a fix. The issue arises when toggling WFS layers on and off at startup: while the visibility settings are correctly saved, a trigger is missing that checks layer visibility upon loading. As a result, when a WFS layer loads, its visibility is not accurately represented.

3D Tiles

The primary objective of creating 3D Tiles in the Delft3D project was to add detailed vegetation models, specifically for bushes and grass, as requested by the Municipality of Delft. While Netherlands3D already includes 3D representations of trees based on the data from BGT (Basisregistratie Grootschalige Topografie) , incorporating additional vegetation types like bushes and grass would support urban planning by providing a complete 3D visualization of green spaces in Delft.

To achieve this, we experimented with several approaches. We initially explored using Unity directly by following the Netherlands3D documentation on adding a simple layer. However, this solution required more advanced C# and Unity skills than could be reasonably achieved within the project timeline. Consequently, we opted to utilize the 3D Tiles support in Netherlands3D as the most suitable approach. This method allows for loading 3D Tiles by providing a tileset URL, enabling them to be rendered directly within the project.

Steps for Creating a 3D Tileset Creating 3D Tiles tileset involves two primary steps:

- **Step 1: Converting 2D Points Data to 3D Models** - We used FME (Feature Manipulation Engine) to convert 2D GIS data for vegetation, specifically bushes and grass, into 3D models in the CityGML format. This process was adapted from the CityGML creation workflow provided by Safe Software, which can be referenced for additional details on creating solitary vegetation objects: CityGML Solitary Vegetation Object Creation Guide. Below is the detailed process to create the CityGML files for vegetation. Any mention of

"Bookmarks" (e.g., "store and remove coordinates" Bookmark) refers to the labelled sections within the FME workbench that streamline tasks by grouping related transformers or processes (as shown in Figure 7 in Appendix D):

1. **Add Source Data Containing Tree Positions** - Load the source data for vegetation positions in FME, sourced from Open Data Portaal Delft. This dataset, consisting of point data, is initially provided in the CRS EPSG:3857 - WGS 84 / Pseudo-Mercator (Projected). For further processing, we clipped and reprojected it to EPSG:7415. This data is used solely for testing purposes, as the actual shrub data will be in polygon format and sourced from the BGT website.

2. **Add Source Data Containing Tree Models** - Import 3D vegetation model into FME to use as reference geometry.

3. **Apply 3D Forcer to the Source Data Containing Tree Positions** - Since the vegetation data doesnt contain z-value, this transformer is used. We used "1" for the elevation value for testing purpose.

4. **Add Attributes** - Using the AttributeManager Transformer, we added information regarding the height and the tree model that we want to use for each tree. Since the client requires bushes and grass to have the same height we used 13.6 meter height and chose one of the available tree model in the workbench.

5. **Prepare Vegetation Positions** - Using the "store and remove coordinates" Bookmark, adjust the vegetation position data to store only the coordinates for placing each model correctly.

6. **Prepare Vegetation Models** - Similarly, use "prepare geometry instance" Bookmark to format the vegetation models to apply them to each tree position.

7. **Merge Position and Model with FeatureMerger** - Using the FeatureMerger tool, combine the position data and model data. This allows each vegetation position to be associated with the corresponding 3D model.

8. **Set Shared Item for Geometry Instance** - Configure the merged models to share geometry instances by following the "set SharedItem (geometry instance)" Bookmark, to handle repeated geometries in CityGML format.

9. **Set CityGML Specific Properties** - This is to ensure compatibility with CityGML standards, such as object type, elevation, and vegetation characteristics.

10. **Define Destination Schema** - Define the destination schema to specify that the output should conform to the CityGML format.

11. **Prompt and Run Workspace** - Generate a CityGML file containing the vegetation data as 3D models ready for further processing. The result of the entire process is a CityGML file containing the vegetation data as 3D models ready for further processing in the CRS EPSG:7415.

- **Step 2: Converting 3D Models to 3D Tiles** - This step also uses FME for writing 3D Tiles.

    1. **Add Cesium 3D Tiles Writer** - Write it to 3D Tiles using the Cesium 3D Tiler, setting the geometry type as cesium_3d_object. We used Cesium ion to check whether the output is displayed accordingly. The writer automatically handles projection.

Outcome of 3D Tiles Integration Despite following the steps described and working closely with the Netherlands3D team, we were unable to load the 3D grass and bushes tiles into the project. We identified the problems while loading the 3D Tiles into the project, mainly misalignments in the Tiles standard and certain server security issues that hindered the process. Due to the time constraint of this project and the actively evolving Netherlands3D platform, we decided to come to a stop. However, we believe this integration will be successful with future developments in Netherlands3D along with its guideline in incorporating 3D Tiles in the project.

Reflections Throughout the 3D tile integration process, we gained insights into the challenges and limitations of working with an evolving platform such as Netherlands3D. One lesson was the importance of documentation, which would provide specific guidance when errors occur. We relied heavily on trial-and-error approaches and ongoing communication with the developers, which resulted in delays in this part of the project. The knowledge of tools, such as Unity and FME, and the adaptability in exploring different methods are essential to working with an actively developing project.

Suggestions for Future Works on 3D Vegetation Based on our experience, we propose the following suggestions for future work on Delft3D and similar projects regarding 3D vegetation. It is crucial to acknowledge that these features would likely be integrated in Netherlands3D in the following years.

- **Evaluate Alternative 3D Tiles Generation Tools** - While FME was used for generating 3D models and tiles for this project, exploring other tools with established 3D Tiles workflows—such as Tyler—may be valuable. However, one should keep in mind that different methods require different types of input data and workflow than the approach that we took.
- **Ongoing Collaboration with Netherlands3D Developers** - The evolving nature of the Netherlands3D project requires close collaboration with the developers. Checking regular updates on new features and documentation will help align project objectives and reduce possible troubleshooting.

By incorporating these recommendations, future users can address some of the current platform limitations and streamline the process of integrating custom vegetation models into Netherlands3D.

3.4 Added Features and Customizations

Replacing buildings with High Definition (HD) models

Some landmarks in Delft are ideally displayed in higher quality than the current Level of Detail (LoD) available in the viewer. To achieve this, certain buildings need to be removed and replaced with custom OBJ models. Below, we provide a quick explanation of how custom OBJs are placed. Individual buildings cannot be accessed as separate GameObjects, which complicates the removal process slightly. The framework loads 3D tiles as `GameObjects`, subdividing them into sub-`GameObjects` for each tile and referencing buildings as items, assigned in the `ObjectMapping.cs` script. In this script, `ObjectMapping` associates buildings with specific objects within a tile by assigning each building an ObjectID. The script checks each building's first vertex to find its unique identifier, helping manage individual structures in the viewer despite the initial lack of direct `GameObject` access. However, for an object to be removed, it needs to be represented as a GameObject.

We can still alter the visibility of particular buildings by accessing their unique BAG (Basisregistratie Adressen en Gebouwen) IDs. In the `ToggleVisibilityForBuildingsWithBagIds.cs` script, we set each building's colour to fully transparent, effectively hiding or showing them in the Unity scene.

Area of Interest (AOI)

Since this project is intended for a certain municipality, it is possible to restrict the movement of the users to a certain area. This is done by specifying the desired region as a polygon in any GIS software and exporting it as a `GeoJSON` file, named 'aoi.geojson'. To apply this area of interest to the project, the aoi.geojson has to be uploaded alongside the build files. More information on the upload can be found in the Delft3D's documentation 2024

Custom OBJ Layers

To allow users to upload custom 3D models in OBJ format, we implemented a feature that enables the loading and visualisation of these models within Delft3D. This capability is especially useful for municipalities to see specific buildings or structures in the 3D environment, such as proposed new developments or detailed architectural models. For more details on the technical setup and code snippets, users can refer to the Delft3D's documentation 2024.

Approach To permanently store the object files and their positioning, they are uploaded to a remote server. This happens through HTTP requests, which are sent to a simple REST API[1]. The server then stores the OBJ file in its local file system and inserts the metadata into the database. Once the object is saved to the server, it loads automatically to each new client session. Figure 5 illustrates the request-response cycle between the client and the server.

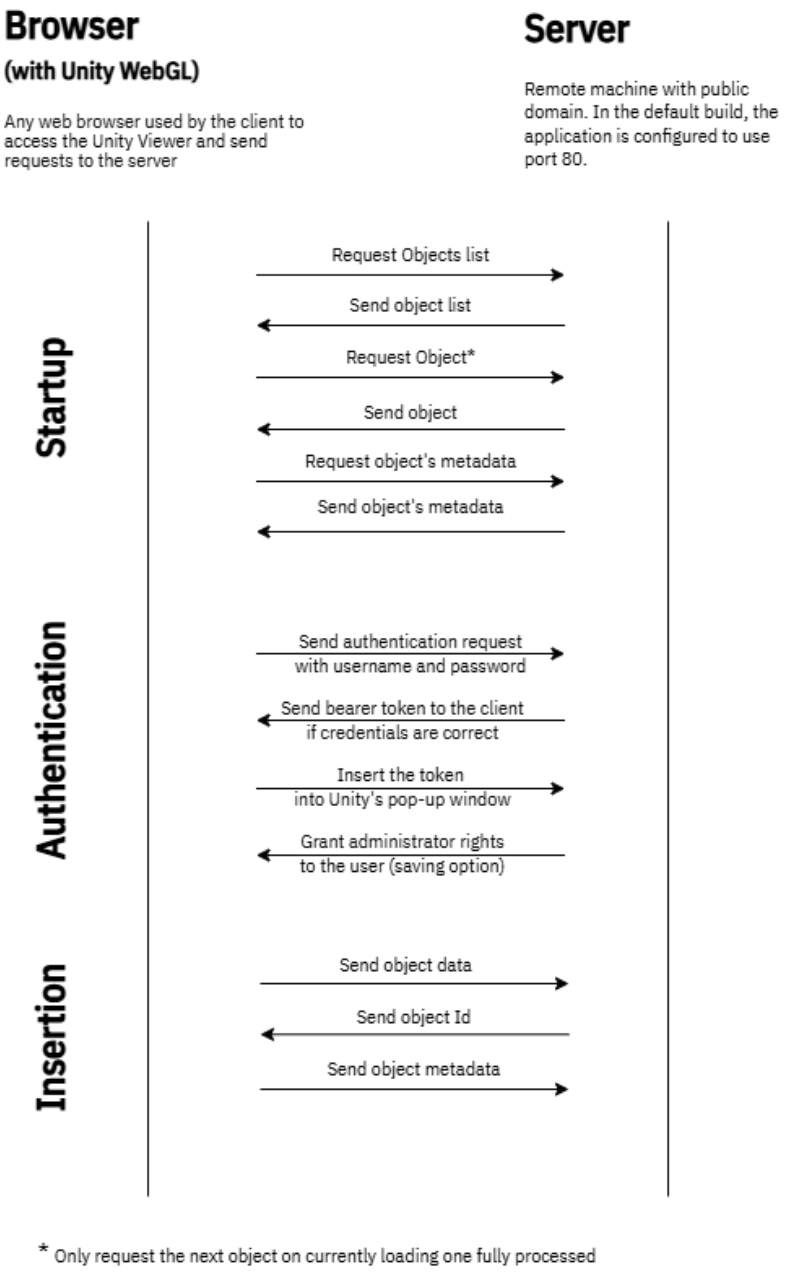


Figure 5: Communication between the client and the server at various points of the process.

Challenges and Limitations During the development of the custom OBJ layer functionality, several challenges were encountered:

- **Server Connection**
  - Establishing a connection from Unity to our server posed initial challenges due to our self-signed HTTPS

[1]REST: `https://en.wikipedia.org/wiki/REST`

certificate, which Unity blocks for security. We addressed this by implementing a bypass script.

  - Distinguishing between the initial model upload and subsequent updates was crucial. Each model is assigned a GUID in Unity, which is replaced by a UUID from our database. This UUID must be returned to Unity to overwrite the original GUID, preventing duplicate model entries.

- **Local File Reading**

  - The original Netherlands3D functionality loads an OBJ by creating a temporary file with its content. This approach was problematic, necessitating a method to retrieve and store the file's content on the server.

  - Selecting the appropriate model for database submission was challenging; although raycasting initially functioned, it failed to select certain surfaces. Ultimately, we utilized an existing function for selecting the current layer developed by the Netherlands3D team.

  - Storing the object's position, rotation, and scale also presented difficulties. We accessed these properties through the attached gizmo and its panel. However, we later had to translate the location to RDNAP (EPSG:7415[2]) coordinates for proper storage and retrieval.

- **Downloading files** - When the built application runs in a browser environment, it is not possible to write directly to the user's file system without explicit permission (and path selection). Therefore, when loading files from the server, they have to be stored in memory. Since the Netherlands3D framework already implements a functionality to read from a file, it only had to be somewhat modified to read from memory. Once this functionality creates the object in the game, the respective metadata is loaded from the server and used to overwrite the object's transformation and attributes.

Functionalities When uploading objects the user can specify if the object is a Masterplan (i.e. a building that does not exist in the present) or its just a higher-definition version of an existing building.

4 Results

This section provides an overview of the accomplishments of the Delft3D project. The primary objectives, categorised under the "Must" requirements, were all met. The project also meets half of the "Should" requirements. The model upload and the 3D masterplans functionality are achieved with the "Object Upload" functionality which we explain in Section 3.4. Table 2 provides an overview of the satisfied requirements.

Unfortunately, the 2D Masterplans and the Data Synchronisation requirement could not be satisfied due to limitations discussed in the respective sections of the report. Furthermore, the most suitable approach to implement these layers is with a WMS request, which is going to be implemented by Netherlands3D Developers in the future. Therefore, our recommendation is to wait for this functionality to be implemented by the Netherlands3D team. The hurdles with these requirements are further discussed in Section 6.1.

As for objectives from the category "Could", the temporal slider function has been redefined to a toggle button depicting either the current state or the future one. Together with the client, we concluded, that this will suit the client's needs more since citizens can view Delft at present or with its planned developments. A slider could be a future implementation as an extra refinement giving the user more control over the temporal aspect.

Water levels have been present from the very beginning depicting the actual heights. We recognise that the inclusion of flood models was one of the client's requests; however, we also learned the importance of filtering such requests. Accepting every demand could lead to irresponsible outcomes, especially with something as critical as flood modelling. For instance, a client might rely on an unverified model for flood safety, only to face real consequences if their home is later affected by flooding. Therefore, we determined that flood models would remain out of scope for this project, keeping the focus of the course in mind.

[2] `https://epsg.org/crs_7415/Amersfoort-RD-New-NAP-height.html`

Category	Requirement	Achieved
Must	Terrain (1)	✓
	Buildings (2)	✓
	Vegetation (3)	✓
	Ground Surfaces (4)	✓
	2D Layers (5)	✓
Should	Model Upload (10)	✓
	3D Masterplans (6)	✓
	2D Masterplans (7)	X
	Data Synchronisation (12)	X
Could	Temporal Slider (9)	✓
	Water Levels (11)	✓
	Underground Utilities (8)	X
Won't	Flood Models (13)	X

Table 2: Requirements and their Achievement Status

5 Maintenance of 3D Delft

As already discussed. Delft3D is an extension of the existing Netherlands3D project. One of the reasons for this was thinking about the maintenance and future developments. This means even after its current development is finished, it can still receive updates from its parent project. These updates can be applied by syncing the Delft3D git repository with its upstream - original Netherlands3D repository. Once the syncing is done, the project has to be built again using Unity. The municipality of Delft will be responsible for taking care of the server and the database.

6 Discussion

Overall, this project can be concluded to be successful as we satisfied the most important requirements from the MoSCoW framework.

6.1 Challenges

2D Masterplans The most straightforward way to implement this functionality would have been to import the 2D masterplans as a WMS layer. However, at the time of development, such functionality was not available in the Netherlands3D framework. It is scheduled for implementation in the first quarter of 2025 by the developers of the original project. Given this limitation, along with the project's time constraints, it was decided not to implement this functionality.

Data Synchronization Loading data in a hierarchical manner proved challenging due to time limitations and constraints within the framework, which affected our ability to achieve optimal synchronization. Due to this in combination with the time limitations, it was decided to not implement this functionality.

6.2 Additional Limitations

Overall the learning curve for this project was very steep. Considering that prerequisites for starting this project were not part of our first year's curriculum. (Some) challenges impacted the project:

- **Software Knowledge** - We initially had no knowledge of Unity, particularly with the specifics of the Netherlands3D framework.
- **Limited Experience with C#** - Our team's experience with C# was mostly none. We faced challenges primarily related to C# syntax, as well as the semantics specific to Unity and Netherlands3D, which required

a steep learning curve.

- **Framework Constraints** - The Netherlands3D framework complexity imposed certain constraints that limited our flexibility. While these limitations often stimulated creativity in problem-solving, they likely extended the duration of each task significantly. Had we built the project from scratch, the project and working on it would probably have been more satisfying. However, this was not possible in the short time frame of the project. We are still positive that we chose to use the existing framework since it will set up the Municipality of Delft with a starting point to build upon and develop further, parallel to Netherlands3D.

- **Lack of Documentation** - The lack of documentation for the Netherlands3D framework further complicated our development process, making it difficult to understand existing functionalities and the project structure. The documentation was constantly updated throughout the duration of our project but was not present from the start of it. It is still actively in development.

- **Time Constraints** - Limited time for development added pressure to our workflow, necessitating rapid problem-solving often times leading to overlooking simple solutions provided by Netherlands3D.

6.3 Positive Aspects

We take pride in several aspects of this project:

- Effective communication with clients, which helped clarify expectations and requirements.

- Regular contact with the developers, allowing us to test their product at a very early phase in exchange for their assistance with our project.

- Providing the Municipality of Delft with a foundation for a digital twin and connecting them with the developers

- Collaborative teamwork, both in groups and subgroups, allowing us to leverage diverse skills and backgrounds.

- Learning to share expertise and allocate tasks based on individual strengths, enhancing overall productivity.

- Navigating an existing framework within a limited timeframe, which required adaptability, resourcefulness and good communication skills.

- Acquiring new skills and combining various areas of expertise, including server setup, database management, REST API requests, creating 3D Tiles, and writing documentation.

The development of Delft3D highlights the potential of web-based urban visualization tools that leverage open data for accessible and interactive city models. By adopting the Netherlands3D framework, we created a platform to visualize Delft's urban features—buildings, vegetation, and water bodies—while meeting the Municipality of Delft's requirements and the academic goals of the TU Delft Synthesis project. Despite the technical challenges, the project has offered us valuable learning experiences. We believe that Delft3D provides a foundation for future improvements and broader use of digital urban models by other municipalities.

ACKNOWLEDGMENTS

We would like to extend our heartfelt gratitude to the following individuals and teams:

- **Mike van Riel**, for his invaluable support with the Netherlands3D framework.

- **Hugo Ledoux and Gina Stavropoulou**, for their insightful guidance and direction throughout the project.

- **Neda Loncaric, Herman Tjesse de Haan, Iulia Sirbu, and the dedicated municipality employees**, whose enthusiasm and commitment to the project were truly inspiring. Your consistent involvement and support made a significant difference.

Thank you all for your contributions!

References

3DGI (n.d.). *GitHub - 3DGI/tyler: Tiling 3D city models encoded in CityJSON*. GitHub. URL: https://github.com/3DGI/tyler.

Amsterdam (n.d.). *GitHub - Amsterdam/3DAmsterdam: Repository for the 3D Amsterdam Unity viewer project*. GitHub. URL: https://github.com/Amsterdam/3DAmsterdam.

Basisregistratie Grootschalige Topografie (BGT) - Kadaster.nl zakelijk (July 2024). *Basisregistratie Grootschalige Topografie (BGT)*. URL: https://www.kadaster.nl/zakelijk/registraties/basisregistraties/bgt.

Cesium ion (June 2015). *Cesium ion*. Cesium. URL: https://cesium.com/platform/cesium-ion/.

Con terra lab (n.d.). *CityGML Create SolitaryVegetationObject with Implicit Representation. FME Hub*. URL: https://hub.safe.com/publishers/con-terra-lab/templates/citygml-create-solitaryvegetationobject-with-implicit-representation#description.

Delft Open Data portaal (n.d.). *Delft Open Data portaal*. URL: https://data.delft.nl/search.

Delft3D's documentation (2024). *Delft3D Documentation*. URL: https://akossarkany.github.io/3DDelft.

GemeenteUtrecht (n.d.). *GitHub - GemeenteUtrecht/3d.utrecht.nl: 3D-stadsmodel voor Utrecht vanuit Unity (game-software)*. GitHub. URL: https://github.com/GemeenteUtrecht/3d.utrecht.nl.

Netherlands3D (n.d.[a]). *GitHub - Netherlands3D/twin: Both a skeleton project for new custom Digital Twins based on Netherlands3D and a generic viewer with selectable features (see website)*. GitHub. URL: https://github.com/Netherlands3D/twin.

– (n.d.[b]). *Projects*. URL: https://netherlands3d.eu/docs/developers/projects/.

Unity Technologies (n.d.[a]). *Learn game development w/ Unity — Courses & tutorials in game design, VR, AR, & Real-time 3D — Unity Learn*. URL: https://learn.unity.com/.

– (n.d.[b]). *Unity - Manual: Unity 6 User Manual*. URL: https://docs.unity3d.com/Manual/index.html.

A List of Requirements

1. **Terrain:** Model the local elevation in the AOI (Area of Interest). This layer is the "ground layer". The resolution of the terrain should be at least 1 value per 10 m.

2. **Buildings:** The built environment in the AOI. The model will have LoD 2.2.

3. **Vegetation**: The flora in the AOI is limited to trees and shrubs. The height, width, and type of the trees should reflect reality as accurately as possible, based on the available data. The data for this will be sourced from the COBRA dataset, which the municipality will provide.

4. **Ground surfaces:** Model the different ground surface types. Types that must be included are roads, bike lanes, vegetation, water, and pavings.

5. **2D Layers:** The application should allow users to project national and local data sets over the 3D model. These layers are either integrated into the application using a database or requested through WMS or WFS services.

6. **3D Masterplans:** Functionality to render 3D models of future urban developments within the current built environment.

7. **2D Masterplans:** Functionality to project 2D plans of future urban developments to the ground surface of the model.

8. **Underground utilities:** Functionality to visualize utilities underground. Given data about underground features, the application should make it possible to render this data for the users.

9. **Temporal Slider:** Adding a slider to navigate between 3D and 2D masterplans across future years.

10. **Model Upload / Model Exchange:** There will be the option to upload building models with a higher Level of Detail (LoD). For now we will focus on importing models in the legacy OBJ format. In case the model is not availabe in OBJ format, users have to convert it using third-party software.

11. **Water levels:** The elevation of water surfaces in the model should reflect the mean water level in the canals.

12. **Data Synchronization:** Upload new data to the model through the open data portal of the municipality. Link the Delft Open Data Portal to 3D Delft by using the ArcGIS API from the data portal, so that when new spatial data is uploaded, it is immediately visible in the 3D Delft viewer.

13. **Flood Models:** Functionality to perform flood modeling in the viewer, either by using existing flood modeling data or through analysis conducted directly within the viewer.

B List of Services from Open Data Portaal

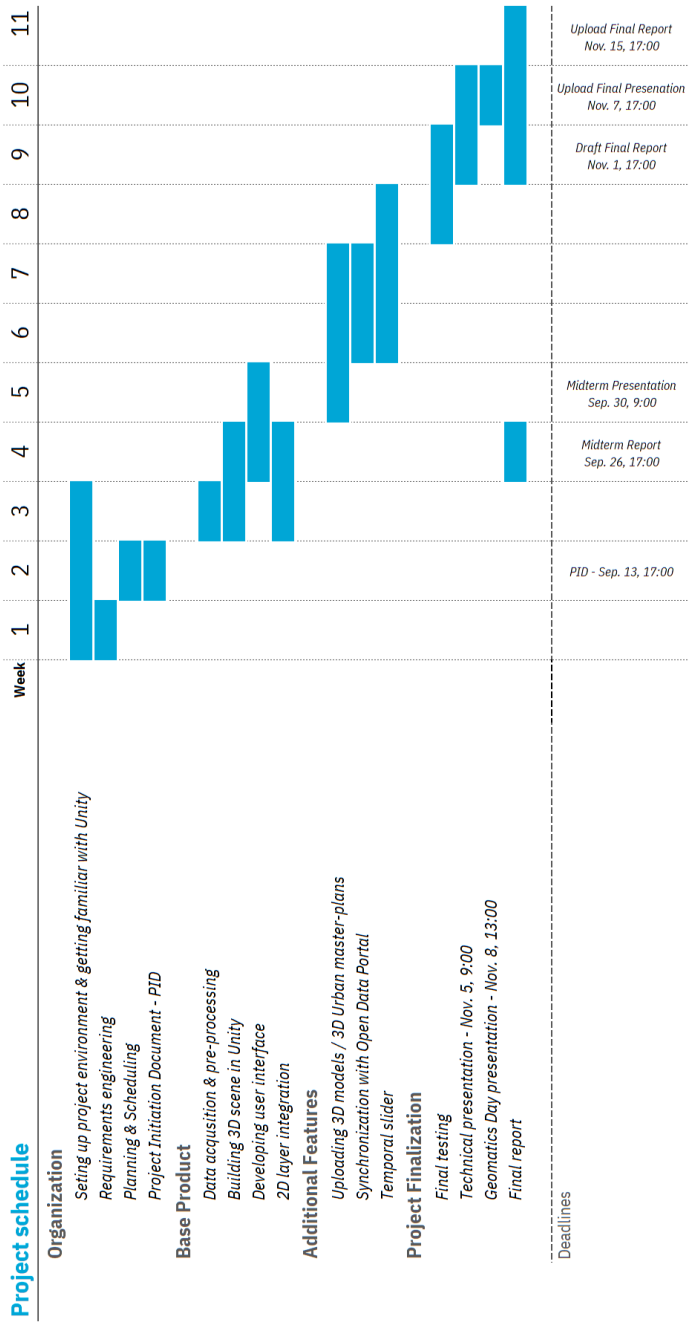
Themakaarten	**WMS**
Luchtfoto PDOK	https://service.pdok.nl/hwh/luchtfotorgb/wms/v1_0?&request=GetCapabilities&service=wms
Infrarood 25 cm PDOK	https://service.pdok.nl/hwh/luchtfotocir/wms/v1_0?&request=GetCapabilities&service=wms
AHN PDOK	https://ahn.arcgisonline.nl/arcgis/rest/services/Hoogtebestand/AHN3_5m/ImageServer
Enkelbestemming PDOK	https://service.pdok.nl/kadaster/plu/wms/v1_0?request=GetCapabilities&service=WMS
Kadastrale kaart PDOK	https://service.pdok.nl/kadaster/kadastralekaart/wms/v5_0?request=GetCapabilities&service=WMS
Hitte monitor-warm nights	Online ESRI ArcGIS (WMS Instruction) — Meteomatics
Hitte monitor-Heat index	
Hitte monitor-Tropical days	
Hitte monitor-Tropica days 2050	
Hitte monitor-Risk	
Klimaareffectatlas	https://apps.geodan.nl/public/data/org/gws/YWFMLMWERURF/kea_public/wms?request=getCapabilities
Oppervlakte temperatuur	geodata.zuid-holland.nl/geoserver/klimaat/wms?service=WMS&version=1.3.0&request=GetCapabilities
Klimaatrisico	data.rivm.nl/geo/ank/wms?
Hitte: kwetsbare gezinnen in warmste buurten	geodata.zuid-holland.nl/geoserver/klimaat/wms?service=WMS&version=1.3.0&request=GetCapabilities
Gevoelstemperatuur N&S	https://tiles.arcgis.com/tiles/nSZVuSZjHpEZZbRo/arcgis/rest/services/Hittekaart_gevoelstemperatuur/MapServer/WMTS/1.0.0/WMTSCapabilities.xml
Klimaat atlas: afstand tot koelte	https://tiles.arcgis.com/tiles/nSZVuSZjHpEZZbRo/arcgis/rest/services/Afstand_tot_koelte/MapServer/WMTS/1.0.0/WMTSCapabilities.xml
Klimaat atlas: bodemdaling 20-50	geodata.zuid-holland.nl/geoserver/klimaat/wms?service=WMS&version=1.3.0&request=GetCapabilities
Aanvullende daling	geodata.zuid-holland.nl/geoserver/klimaat/wms?service=WMS&version=1.3.0&request=GetCapabilities
Open data Delft	
Bedrijvengebieden	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/Bedrijvengebieden/FeatureServer/0/query?outFields=*&where=1%3D1
Wijken	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/wijken/FeatureServer/0/query?outFields=*&where=1%3D1
Buurten	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/buurten/FeatureServer/0/query?outFields=*&where=1%3D1
Monumenten	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/Monumenten/FeatureServer/1/query?outFields=*&where=1%3D1
Kunst in openbare ruimte	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/Delft_compleet_kunstwerken/FeatureServer/0/query?outFields=*&where=1%3D1
Musea	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/Musea_in_de_gemeente_Delft/FeatureServer/0/query?outFields=*&where=1%3D1
Speelplekken	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/Speelplekken/FeatureServer/0/query?outFields=*&where=1%3D1
Evenementen locatie	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/evenementenlocaties/FeatureServer/0/query?outFields=*&where=1%3D1

Bomen in Beheer door gemeente Delft	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/Bomen_in_beheer_door_gemeente_Delft/FeatureServer/0/query?outFields=*&where=1%3D1
Infraroodmeting wegen	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/Infraroodmeting_wegen/FeatureServer/0/query?outFields=*&where=1%3D1
Afvalbakken	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/afvalbakken(bacdb4f45d5b4239875ae8b093ee372a)/FeatureServer/0/query?outFields=*&where=1%3D1
Beplantingen	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/beplantingen/FeatureServer/0/query?outFields=*&where=1%3D1
Parkverharding	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/parkverhardingen/FeatureServer/0/query?outFields=*&where=1%3D1
Wateren	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/wateren/FeatureServer/0/query?outFields=*&where=1%3D1
Grassen	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/grassen/FeatureServer/0/query?outFields=*&where=1%3D1
Hagen	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/hagen/FeatureServer/0/query?outFields=*&where=1%3D1
Milieuzone	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/milieuzone/FeatureServer/0/query?outFields=*&where=1%3D1
Buurtpreventie	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/Buurtpreventieteams/FeatureServer/0/query?outFields=*&where=1%3D1
Strooirouters grote strooier	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/Strooiroute_grote_strooier/FeatureServer/0/query?outFields=*&where=1%3D1
Strooiroute kleine strooier	https://services3.arcgis.com/j07voPd56xoB4c87/arcgis/rest/services/Strooiroute_kleine_strooier/FeatureServer/0/query?outFields=*&where=1%3D1
Atlas natuurlijk kapitaal	https://www.atlasnatuurlijkkapitaal.nl/kaarten
Diverse WMS services	

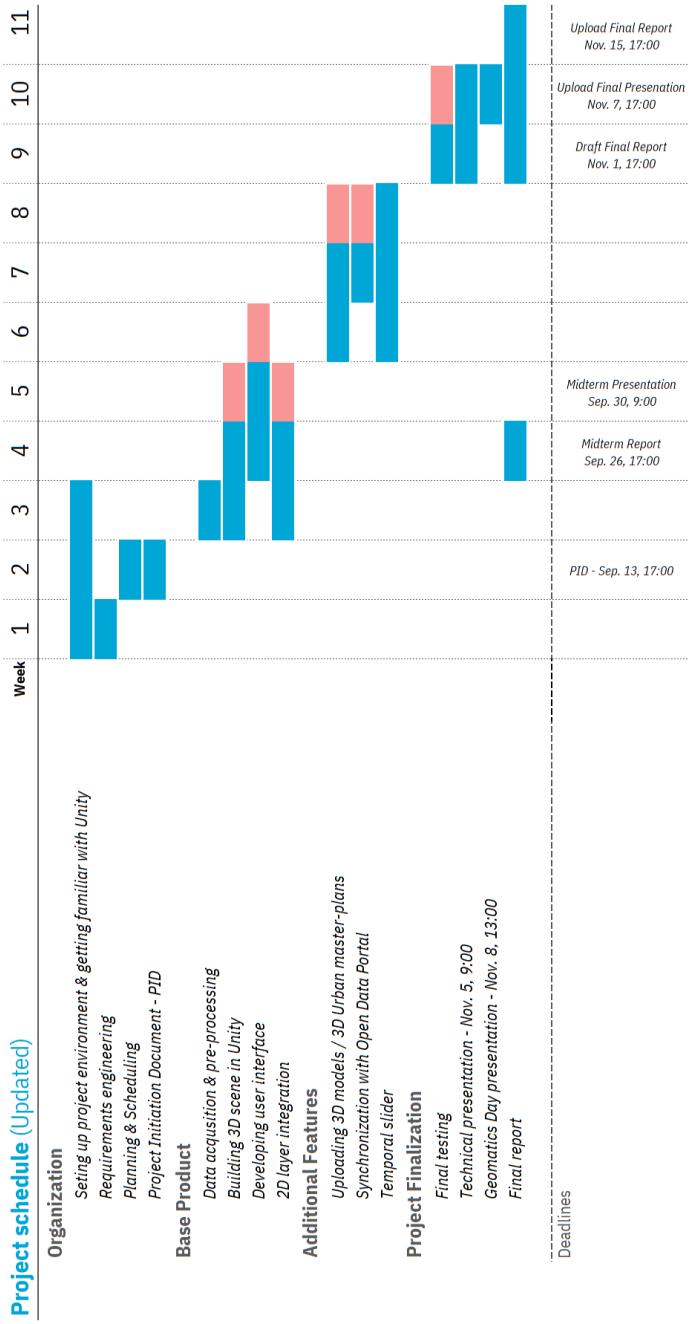
C The WFS layers which The client would like to remove from Delft3D

WFS Layer Name	**Description**
Bedrijvengebieden	Business areas and industrial zones in Delft
Stembureau gemeente Delft 2024	Polling stations in Delft municipality for 2024 elections
Musea	Museums and cultural institutions in Delft
Kunst in openbare ruimte	Public art installations and artwork locations
Evenementen locaties	Event locations and venues throughout Delft
Buurtpreventieteams	Neighborhood watch team locations and coverage areas
Strooiroute grote strooier	Main salt spreading routes for large spreaders
Strooiroute kleine strooier	Secondary salt spreading routes for small spreaders
Infraroodmeting wegen	Infrared measurements of road conditions
Wateren	Water bodies and waterways in Delft
Grassen	Grass areas and green spaces
Parkverhardingen	Paved areas within parks and public spaces

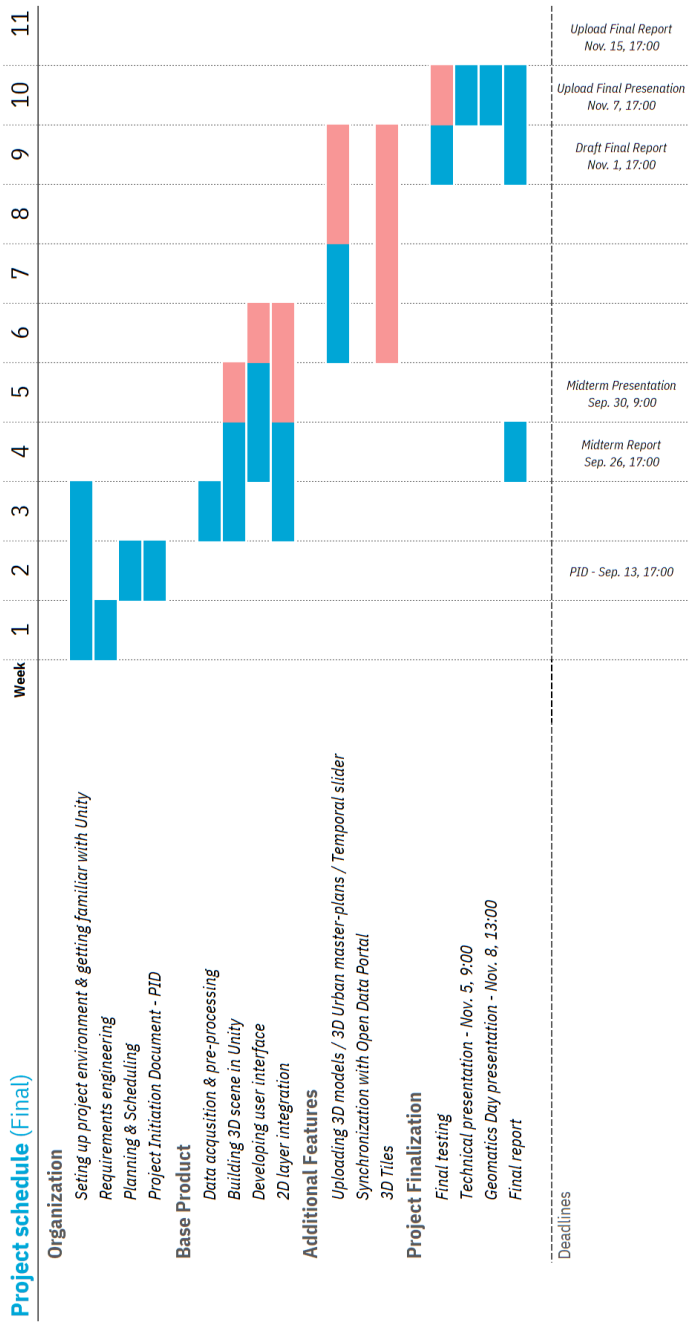
D Additional Figures



(a) The schedule at the start of the project



(b) The mid term updata of the schedule



(c) The actual schedule at the start of the project

Figure 6: Scheduling of the project at various points.

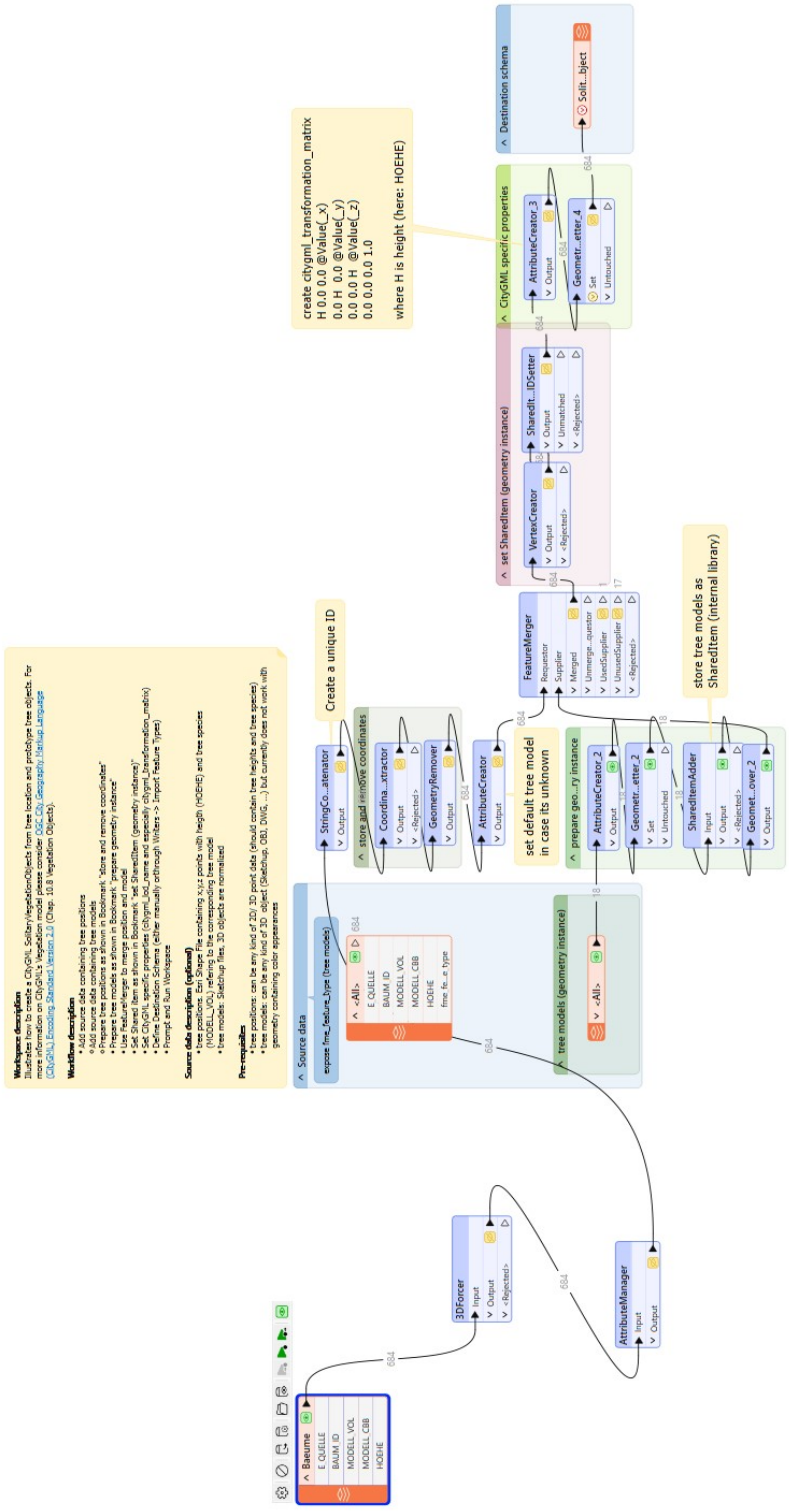


Figure 7: FME Workbench of Step 1 to Create 3D Tiles.