TUTORIALS
Development Setup & Adding new architecture models

HUVis – Hybrid Urban Visualization

Abstract

This document describes how to setup the HUVis development framework with all its components. Furthermore, it gives detailed insight in how to implement a new architecture model into the existing framework and integrate it into the latest HUVis Unity packages.

Dominic Bosch

Contents

Introduction

The HUVis project gathers partners such as the University of Basel, the Baudepartement Basel-Stadt and Vanamco, as an industrial partner, to create a novel approach for city planning. Its aim is to provide a framework that enables the comparison of proposals from different architects to a certain task in city development. The main task is the comparison of all models from several predefined viewpoints against the current situation by toggling quickly between them. The requirement of having the same surface colour on all the models will allow the city developers to eyeball the volume occupied by the different proposals and also provide a fair comparison basis. Additionally, two view modes will allow for different levels of immersion and information. One mode provides a desktop based high-resolution output, enriched with key-figures which describe the currently viewed proposal. The other mode has to empower the user to dive into the virtual world that is generated during this project. This is achieved by adding virtual reality glasses to the technology stack of the HUVis framework.

The tasks for the second phase of this project were to sustain and document the HUVis application architecture. During the time of this second phase, the decision was made to update to the latest software, to redesign the architecture and to replace the OSVR glasses with the latest technology by upgrading to the Oculus Rift in December 2016. The framework is now setup in a modular approach with independent building blocks that can be easily pulled together to the users wishes. This makes the assembly and deployment of a certain scene very quick. The addition of new models needs more care and time because they often come in any arbitrary format, scaling, orientation and transition. The documented tutorial provides a step-by-step guide on how to create a new Unity package out of a new model.

The most important resource for this project is the bitbucket repository which contains all important files to get the application architecture running, except for the required software. The repository can be found here: https://bitbucket.org/hpcunibas/huvishpc.git

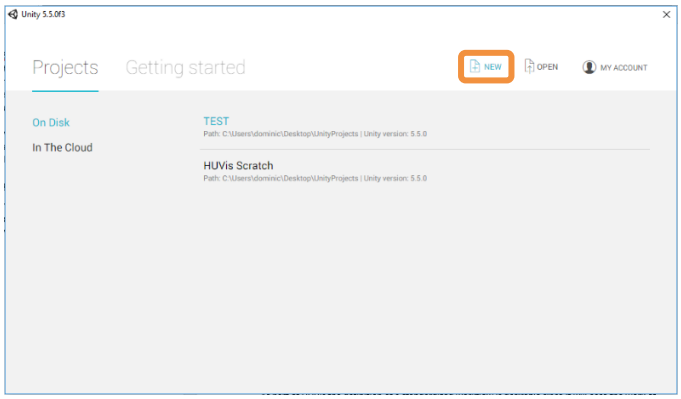
Assembling the HUVis development framework

The HUVis development framework consists of several packages, which are in fact building blocks, that need be imported into Unity in order to further develop the current framework. Instructions on the usage of the required software Unity & SketchUp have already been provided in the preceding bachelor thesis [Dae]. Therefore, this documentation will assume knowledge in how to setup the required terrain which is covered in the thesis. This tutorial sets the focus on the assembly of the available packages from the bitbucket project repository [Bit]. This will lead to a clean state from where parts can be changed, replaced, enhanced or added in order to create a new application from the existing building blocks.

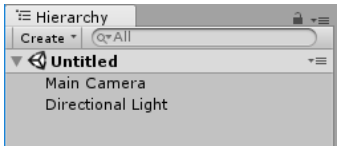
Base Scene

The base scene consists of terrain and directional light (in three forms) only. This is the most basic part of the HUVis application and defines the field of action. Future updates could replace all other parts and still use all the work that has been done in creating the terrain and defining the lightning with the best outcome. This basic block contains the terrain work and the skybox that has been done during the course of [Dae]. Three different light sources have been chosen since they seemed to yield the best lightning results for the exported applications.

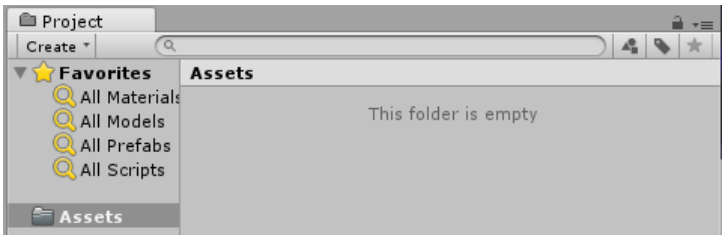
To start from scratch, it's best to start a new project in Unity:



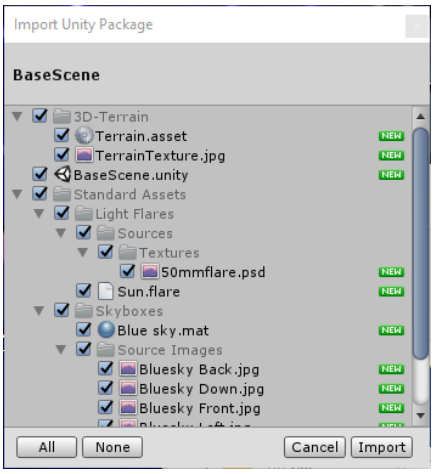
Choose an appropriate name, select 3D and create it. This will provide you the default setup of Unity with a default scene named `Untitled` in the `Hierarchy` window, which contains the `Main Camera` and `Directional Light`:



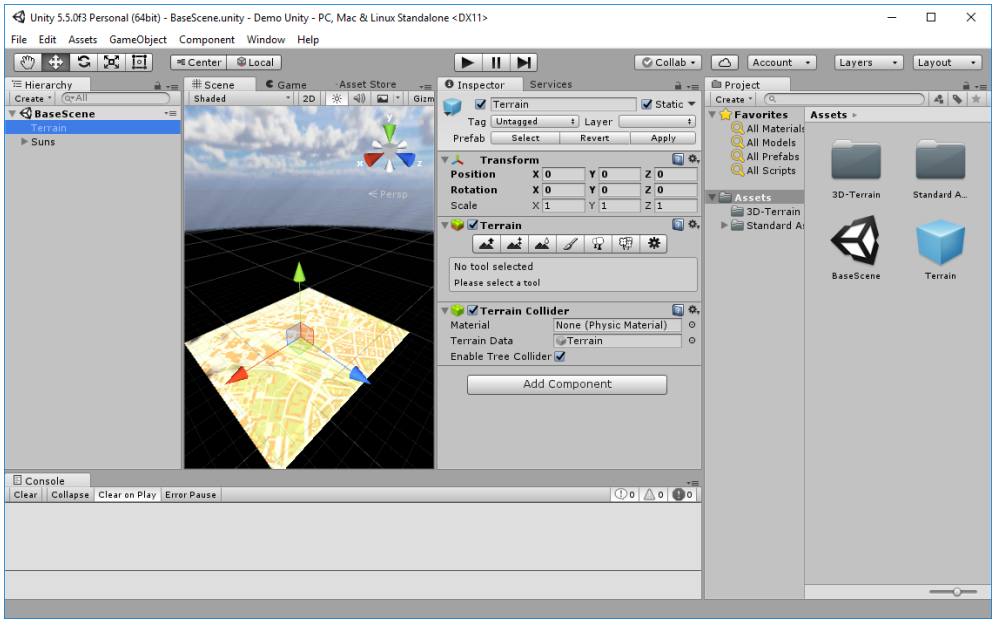
Pull the file `BaseScene.unitypackage` from the folder `Packages` in [Bit] into the `Project` → `Assets` window:



This will open up the following import dialog:



Press Import and you will see all the imported files in your `Assets` in Unity. Drag and drop the `BaseScene` into the `Hierarchy` window and remove the scene `Untitled`. This will give you this basic Unity setup:



Congratulations, you went through the basic steps of importing a Unity package and are ready to assemble the rest of the required framework on top of this base scene. You have now the map and terrain of the project area together with some lightning available to you.

Some project specific adjustments are important to be made at this stage. These are points that have to be considered in each Unity project and are worthwhile looking at in this stage. This will save the newcomers some time searching for the settings. Go to `Edit` → `Project Settings` → `Quality`, then in the Inspector:

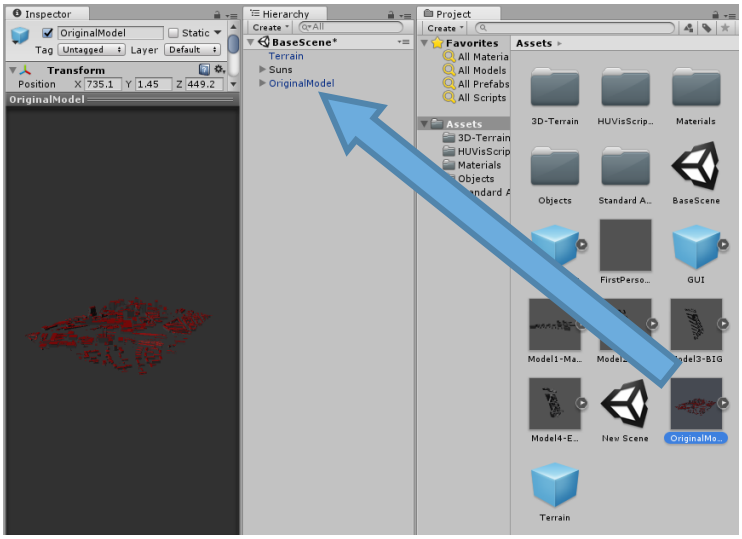
- Increase the `Shadow Distance` to 1000. Since we intend to create an application that allows flying over the whole model, it should be possible to see shadows from a far distance.
- The previous adjustment will cause increased usage of computational resources in order to calculate the shadows. Depending on the kind of final application you are aiming for, you want to adjust the `Cascade splits` which defines how accurate the shadows are as a

function of their distance to the camera. Since the shadow does not need to be very accurate in far distances (as the previously defined 1000 meters), it might make sense to decrease the size of the cascades one and two, and therefore increase the far off cascade number three.

Original Model

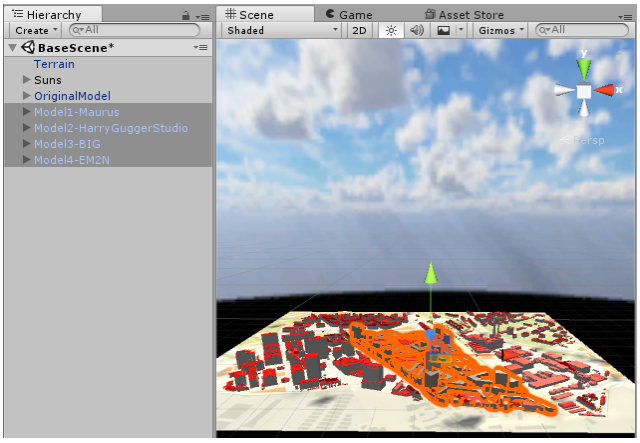
The original model is the status quo of the city, over which new architectural models are planned. In the current, setup the distinction between the original model and the new architectural models can be easily done through the colour of the surface and the fact the original model has red roofs, while the new ones don't. It holds a toggling backbone script which shows or hides certain parts of the original buildings, depending on the model that is shown.

Adding the original model to the HUVis development framework is as simple as importing the unity package `OriginalModel.unitypackage` into the `Assets` folder of your project and then dragging and dropping the prefab called `Original Model` into your scene:



Architecture Models

Currently there are four different models available `Model[1-4].unitypackage`. These models are optional and it is possible to add as many or few of them as desired by the current task. In the available unity packages, the models are each assigned a key (`F2`-`F5`) that will show the model and hide the necessary buildings from the original city model. Adding them to the `BaseScene` is working in the exact same way as for the original model in the previous step. Import one after the other into the project's `Assets` folder and then drag and drop the contained prefabs into the `BaseScene`:



Camera

Every scene needs a camera in order to work properly. In the current HUVis codebase there are three camera packages to choose from. Beware that it is only possible to have one camera per scene!

GUI

Build the Scene

Import a new architectural model

This is an example of how to import a new proposal from an architect into the HUVis development setup in Unity for the model from the EM2N architects. For a trained developer, it takes about 15 minutes to add a new model to the whole HUVis framework.

As part of HUVis the definition of a standardized workflow is desirable since it will ease the work of aligning all the models together into one Unity scene. Such a standard is defined fairly straight forward by requiring any imported model to:

- have units defined in meters,
- be scaled 1:1 with the reality,
- be enriched with global geo coordinates and
- to be in any of these formats: `.fbx`, `.dae`, `.3ds`, `.dxf`, `.obj`, or `.skp`, experience has shown that during export it makes sense to use the ASCII format whenever possible since binary format lead more than once to failed import.

However, the available models from sources such as state departments or open data resources are likely to not adhere to this definition, be it because of missing technology or information. Therefore, it is necessary to cope for this lack of standardization that might be encountered whenever a HUVis package is updated or added.

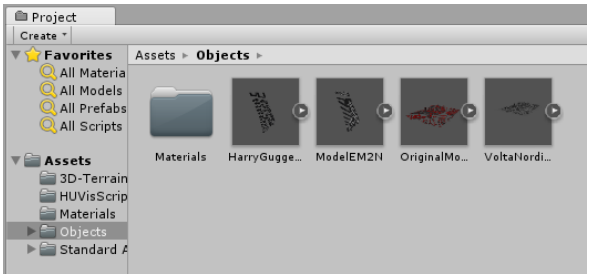
http://www.meshconvert.com/ up to 15 MB

http://www.greentoken.de/onlineconv/ seems not to work a lot

Windows 10 Built-In 3d Builder

Transform into Unity capable format such as ModelEM2N.obj.

Pull into `Assets->Objects` folder

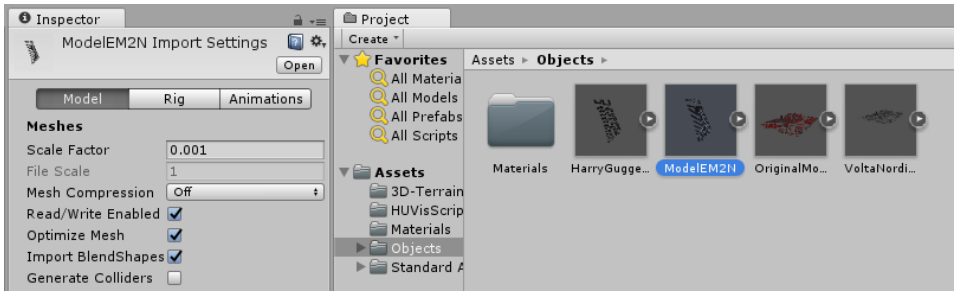


Scaling

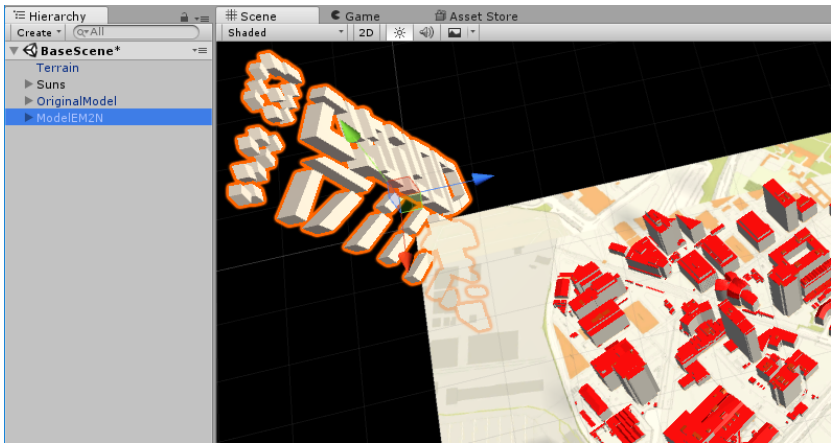
Preferred way: model already aligned with city, Import just works out.

If not, the model needs to be scaled, oriented and transposed

A good start for the whole transformation of the model could be to adjust the import scaling factor, which can even be applied after the import. First the imported model needs to be selected in the Project's folder `Assets->Objects` where it was imported to. Then the `Scale Factor` can then be set in the `Inspector`. When a different base unit was used in the imported model, e.g. millimetres instead of metres, applying a `Scale Factor` of 0.001 scales it into the correct range.



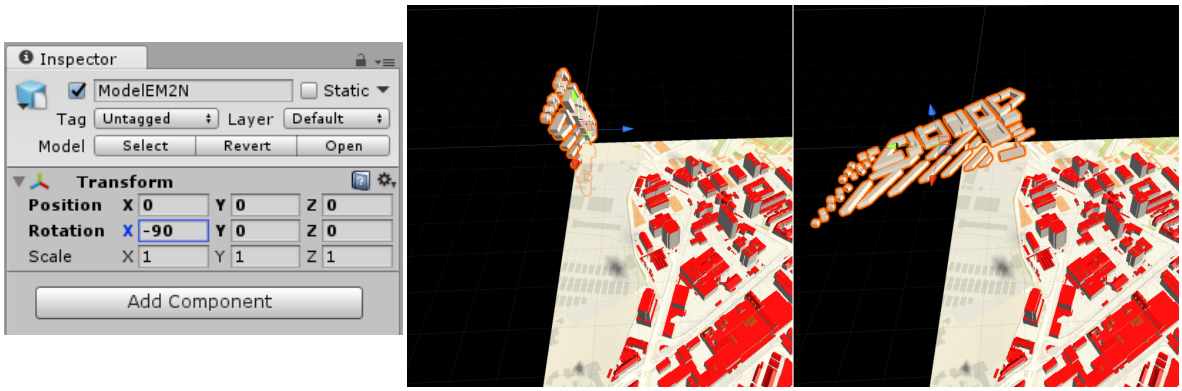
After this, the model can be drag and dropped onto the scene (`BaseScene` in figure below) in the `Hierarchy` window and as a result it gets rendered into the `Scene` window which holds the 3D representation of the current scene:



Should the model still be scaled wrong, it is a good idea to use a 3D modelling application and align the new proposal with the existing virtual world. This can for example be done in SketchUp. After the scaling of the model it of course has to be imported into Unity again, following the previous steps. Using another application makes sense because Unity's emphasis is on building 3D applications, rather than 3D models and therefore, in Unity it is not so easy to align models. The major reason behind switching to a 3D modelling application is the ability to actually measure distances, which is not possible out of the box in Unity. In SketchUp it is possible to measure the distance within the original model and the new one and therefore calculate a scaling factor, which is a solid basis for accurate alignment. One way on how to scale models in SketchUp is described in the next section "Scaling models in SketchUp".

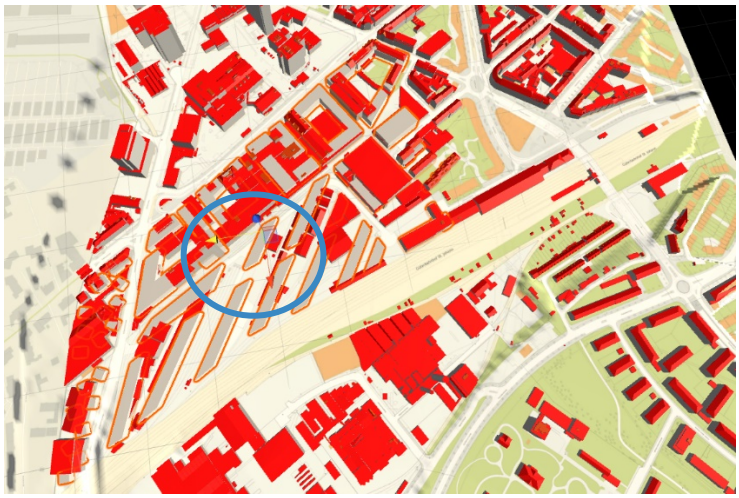
Rotation

It might be that the imported model is aligned along a different axis than the original model. This can be compensated by changing the transform rotation values in the Inspector after selecting the model in the Hierarchy:

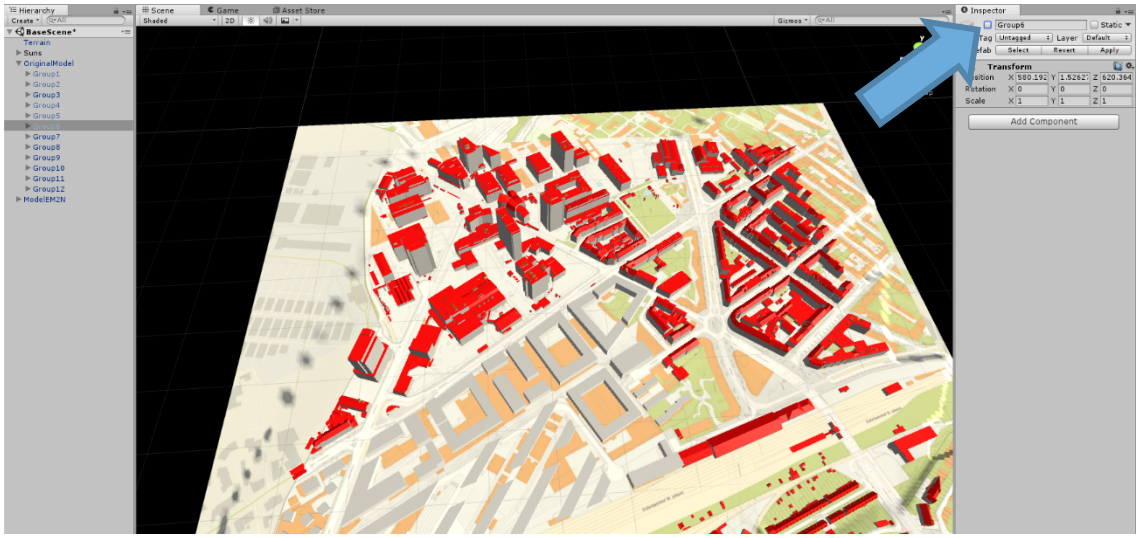


Translation

Relative position information might not be available after the import. As a consequence, the model needs to be positioned manually. A first rough approximation of the correct location of the model can be done by pulling the model with helps of the arrows (shown in the figure below) into the right position within the original model.



Then the groups of the original model which overlap with the new model need to be hidden by selecting them and unticking the `active` flag.



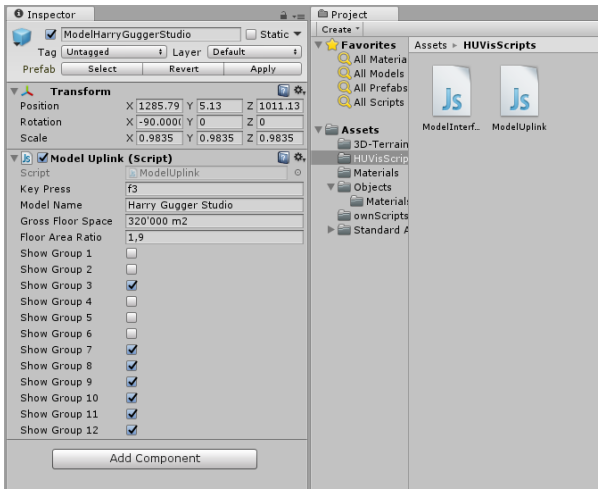
After this, the new model can be more precisely aligned with the map underneath it and by inspection of touching points with other buildings. With the `Position` attributes of the `Transform` dialog in the `Inspector` window, this positioning can be defined very precisely, compared to when using the mouse to pull the model around.

Model Toggle Interface

The different models are going to be toggled through by the user during the demonstration. Therefore, the new model needs to be registered within the HUVis framework. This is fairly straight forward by drag and dropping the `ModelUplink` Script from the folder `Assets->HUVisScripts` onto the model object in the `Hierarchy` tree. The model object needs to have a child called `ModelGroup` under which all to be rendered meshes are located, otherwise the script will run into an error.

The `ModelUplink` script has predefined attributes that require editing:

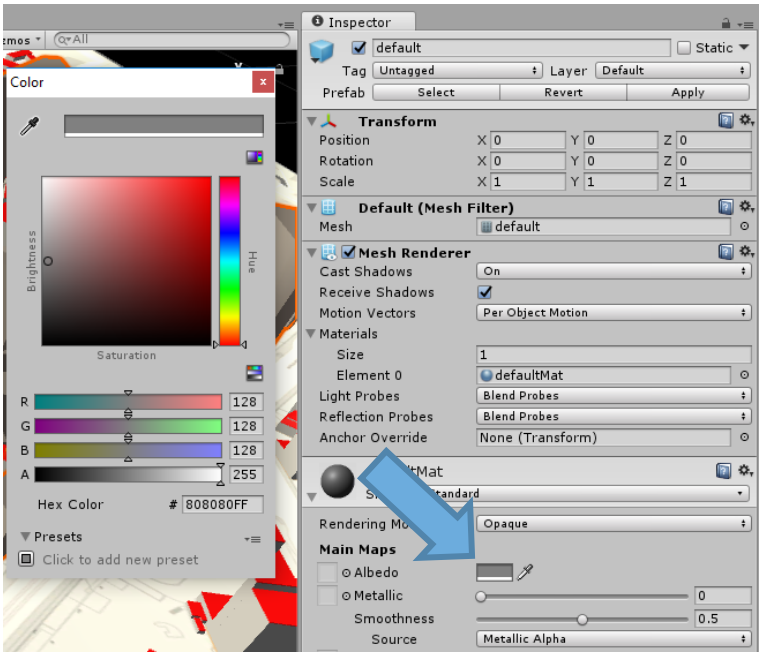
- `Key Press`: the button that shows this model
- `Model Name`: Will be shown in the GUI and needs to be unique across all models
- `Gross Floor Space`: model key figure that will be shown in the GUI
- `Floor Area Ratio`: model key figure that will be shown in the GUI
- The groups of the original model that have to be shown or hidden



The original model is grouped into certain areas that can be shown or hidden altogether. This makes toggling between the different models fairly straight forward. The tick boxes in the `Model Uplink` dialog above correspond to the following grouping:

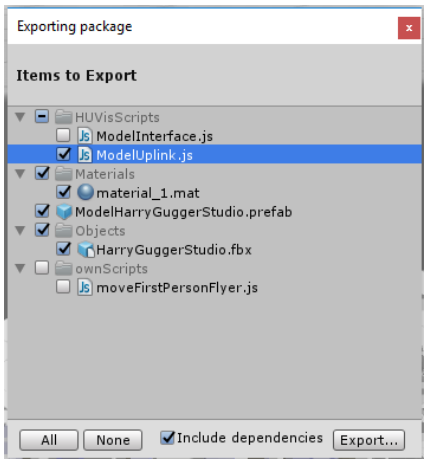
Unified surface Colours

Since all models need to look the same to provide a fair comparison basis in architectural challenges, the textures of the buildings need to be the same: `#808080FF`. This can usually easily be changed in Unity by looking for the material attached to the meshes and changing the `Main Maps` attribute. This usually changes the colour for all meshes in one step as long as not several materials are used in one model.



Export the aligned Model

Create a prefab in the `Assets` folder of the project and named it meaningful. Drag and drop the model from the `Hierarchy` dialog onto the freshly created prefab. Right-Click on the prefab and select `Export Package…` you will see all the objects that will be included in the `.unitypackage` file:



Click on `Export…` if you agree to all the objects included and Unity will generate a `.unitypackage` file that can be reused in a modular fashion. This is a compressed file format and well suited for collaboration.

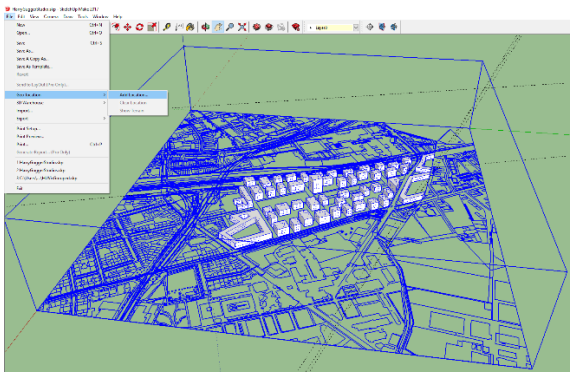
Scaling models in SketchUp

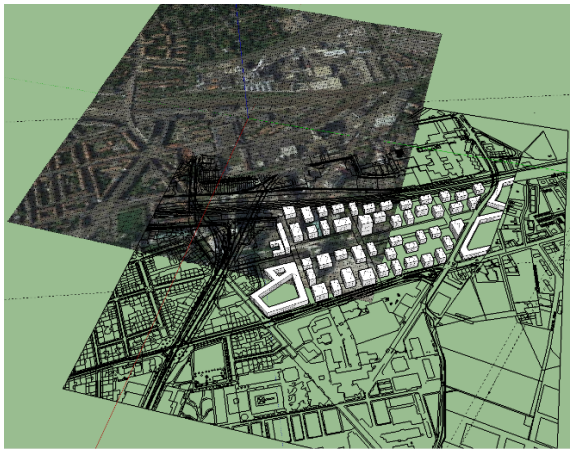
As part of HUVis the definition of a standardized workflow is desirable since it will ease the work of aligning all the models together into one Unity scene. Such a standard is defined fairly straight forward by requiring any imported model to:

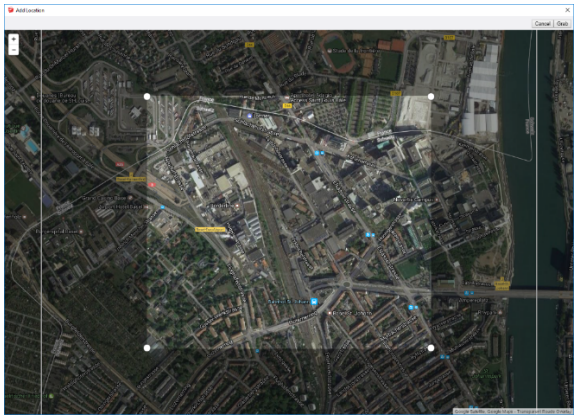
- have units defined in meters,
- be scaled 1:1 with the reality,
- be enriched with global geo locations and
- to be in the **COLLA**borative **D**esign **A**ctivity (COLLADA) format with extension .dae

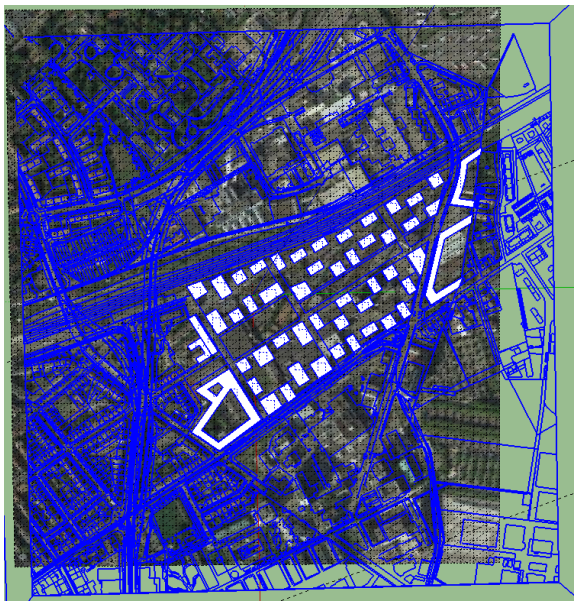
However, the available models from sources such as state departments or open data resources are likely to not adhere to this definition, be it because of missing technology or communication issues. Therefore, it is necessary to cope for this lack of standardization that might be encountered whenever a HUVis package is updated or added.

Geo location for relations









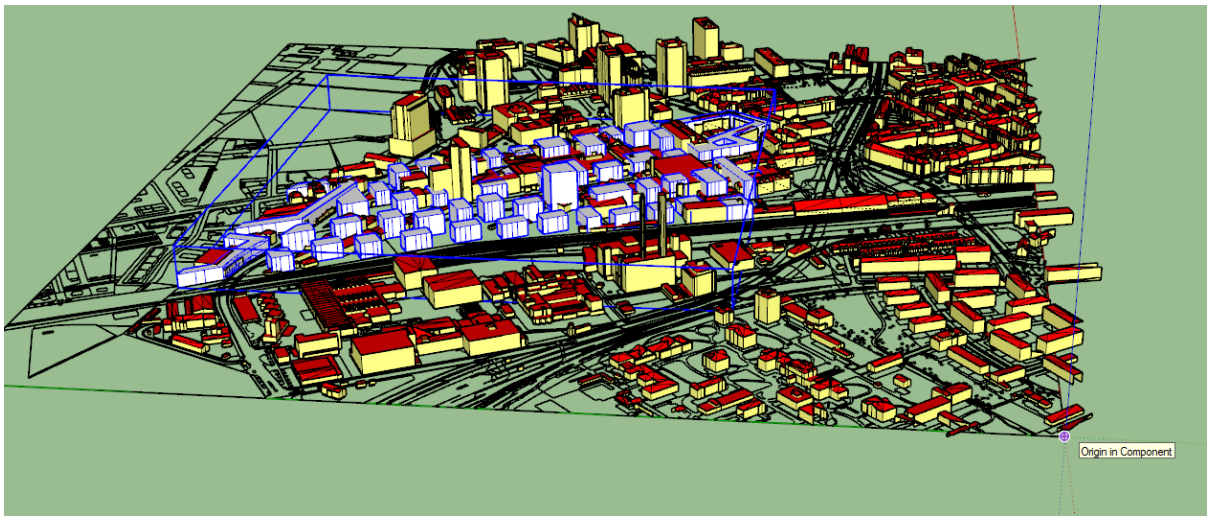
In Sketchup:
#############
small imported model. station length: 2.31
in mapped 1:1 surface: 201.57
scaling:87.25974

```
after: 56.42
to: 49.3
scaling: 1.1444
new scaling: 1.1444 x 87.25974 = 99.861958

59.76
59.62
scaling: 1.00234
new scaling: 99.861958

In Unity:
#########
Transform:

Position: X=1286.85, Y=6.68, Z=1012.5
Rotation: X=0, Y=0, Z=0
Scale: X=98.55, Y=98.55, Z=98.55
```

Transformation of model consisting of 2D planes (Stadtmodell TerrainCut original scale.dae) into 3D model required for performance reasons: 8 hours. Requires definition of meaningful components that belong together. And finally the scaling and trasnformation paramters have to be found in order to get the model into Unity.

From Sketchup export into fbx. Combining several objects into components causes meshes to be generated which reduce computations for the virtual reality drastically.

SkyBox



References

[Dae] Dähler, M.: Augmented Reality: Immersionsszenarien, Technologien und Fallstudie. Bachelorarbeit. 2014

[Bit] Bitbucket project repository https://bitbucket.org/hpcunibas/huvishpc.git

[Gru] Grundbuch und Vermessungsamt http://www.geo.bs.ch/