CHALMERS

Virtual Reality Application User Interface

Design and Implementation of controls for navigation in a Virtual Reality application

Master of Science Thesis in Interaction Design

MIKAEL NILSSON

FREDRIK WENDT

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2010

Virtual Reality Application User Interface

Design and Implementation of controls for navigation in a Virtual Reality application

MIKAEL NILSSON

FREDRIK WENDT

Examiner: MORTEN FJELD

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering

Göteborg, Sweden June 2010

ABSTRACT

As the graphics processing power of home computers increase Virtual Reality applications become more and more viable for a number of uses. User interfaces in many such applications today require the user to have a fairly high degree of skill. This report describes a project creating navigation controls and a graphical user interface for a Virtual Reality application aimed at users at the beginner level. The report details the design of a number of tools available in the application as well as results from user testing with prototype versions. The result of the report is a combination of the prototypes tested to form a fully functioning user interface.

PREFACE

This thesis was written for Chalmers University of Technology, Göteborg, Sweden as part of the Master Program Interaction Design. The thesis was performed by Mikael Nilsson and Fredrik Wendt at Dynagraph, Göteborg.

Examiner and supervisor at Chalmers for this project was Morten Fjeld at the department of Interaction Design. Supervisor at Dynagraph was Carl Johan Andhill.

We would like to thank the people who participated in our user tests as well as any other who have given input.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

3D	Three Dimensional
2D	Two Dimensional
VR	Virtual Reality
GUI	Graphical User Interface
IMGUI	Immediate Mode Graphical User Interface
DOF	Degrees of Freedom
LMB	Left Mouse Button
RMB	Right Mouse Button

1 INTRODUCTION

As the graphics processors found in the average office computer increase in computational power, the use of Virtual Reality (VR) applications increases, both commercially and non-commercially. The interfaces and controls of these applications may, however, often require the user to be more adept than the average intended user.

In order to address this issue, this project was started at the initiative of Dynagraph, a company based in Göteborg. A big part of Dynagraph's area of expertise is the building of VR models and environments for different uses.

1.1 BACKGROUND

One of the services offered by Dynagraph is the ability to look at construction projects before they are built by using a detailed three dimensional (3D) model and viewing it in a VR application. This gives architects and financiers a chance to see the impact different choices will have on the project itself and the surroundings.

Dynagraph has noticed that only a small part of the VR application used for their models is utilized by their customers and that the application in some cases is hardly used at all after purchase. The two main reasons for this are believed to be:

- **Availability**: The system requirements for the application are too high. For instance, the application requires a fairly good graphics processing unit not available in many computers.
- **Controls**: The controls are hard to understand and handle for users not accustomed to similar software.

1.2 PURPOSE

The purpose of the project is to, in more detail, identify the problems with an existing control solution and use this analysis as the basis on which to develop an improved version of the control system. The project entails introducing a number of these alternative solutions as well as evaluating how well they achieve the target goal. The final goal of the project is to present a fully functioning prototype of the interface.

1.3 DELIMITATIONS

Since priority has been placed on developing a fully functioning interface, certain solutions that were deemed not to have been thoroughly tested or too time-consuming to implement were not investigated more closely. Some of the techniques behind these solutions were still reviewed for inspiration.

Due to the nature of the intended user of the VR models, alternative input devices - e.g. SpaceBall or Wii Remote - were not considered, nor were alternative output devices, such as 3D glasses.

This project has been performed in parallel with the design and implementation of another part of the interface. Navigational aids such as maps have been addressed by that part of the implementation and are therefore not addressed by this project or in this report.

1.4 METHOD

The interface is implemented in Unity, a content centered development environment for building games. The efficient implementation tools provided by Unity make it possible to create simple working prototypes directly, removing the need for other prototyping methods such as paper prototyping.

The first step of the design phase is to look at the existing solution and identify the problems found in the interaction of the system. After that the project is performed iteratively, with a prototype created and tested each iteration. The prototypes are designed to test control schemes found in relevant literature, or to test changes made to the previous prototype to establish the best solution to a specific design problem.

1.5 REPORT OVERVIEW

This report is divided into seven chapters. The first chapter is an introduction to the problem and the problem background as well as why the project has been started. The introduction also gives a short description of the design method used.

In the second chapter, *Theory*, we present the theoretical background needed to understand this report and the design project behind it. The chapter presents theoretical background in *Interaction Design Methods*, the *Unity Game Development Environment* and *3D Navigation*. The third chapter is a simple list of requirements, informally stated.

The fourth chapter, *Method*, presents the design method used in this project in more detail. It is divided into *Navigation* and *GUI* and explains the structure of each part of the user interface as well as test results reached when developing it.

In the fifth chapter the results achieved in the project are presented. They are often presented as references to the relevant design versions in the *Method* chapter. In the sixth chapter, *Discussion*, we discuss the results reached. The seventh chapter is a list of references.

2 THEORY

In this section an overview of the theoretical background of the project is presented. It entails an overview of interaction design methods, the game development environment Unity and 3D navigation.

2.1 INTERACTION DESIGN METHODS

There are a great number of methods and techniques available when designing a user interface. The methods used in this project are presented in this section. The reasons for choosing design methods vary, but include time constraints, the availability of test persons, budget and implementation methods.

2.1.1 PROTOTYPING

Prototyping is a technique used in interaction design to test usability and design properties of a user interface before creating a full implementation. It can be performed by creating working, but simple, versions of the interface or by creating mock versions of the interface with simpler means, e.g. paper prototyping [1].

2.1.2 TASK ORIENTED INTERVIEW WITH TALK-ALOUD

User interviews serve as one of many important tools in interface design as they help the designer discover difficult areas of the interface from the perspective of the user. The user gets a number of specific tasks and a small general walk-through of the software. After a brief interview the test starts. The user then tries to solve the different tasks without help from the observer or the interviewer. If the test person has questions or comments he is encouraged to speak his mind. The comments provided can involve the user's intentions and what the user is trying to do as well as what he is doing that very moment.

2.1.3 COGNITIVE WALKTHROUGH

Cognitive walk-through is a method used to identify usability issues by focusing on specific tasks. The method involves taking a set of tasks and performing a task analysis for each one of them. If there are many ways to perform the task this should be included in the task analysis as well as the different steps involved in performing the task. For each task there are four different questions that should be answered [2]:

- Will the user try to achieve the effect that the sub-task has? Does the user understand that this sub-task is needed to reach the user's goal?
- Will the user notice that the correct action is available? E.g. is the button visible?
- Will the user understand that the wanted sub-task can be achieved by the action? E.g. the right button is visible but the user does not understand the text and will therefore not click on it.
- Does the user get feedback? Will the user know that they have done the right thing after performing the action?

2.1.4 HEURISTIC EVALUATION

Heuristic Evaluation is a method which aims to identify weak design and usability problems in a specific user interface. It is beneficial to use heuristic evaluation several times during the design process since issues can be eliminated before each release making other issues easier to detect in the following evaluation. A heuristic evaluation of the software should be done by more than one person, since persons often discovers different usability problems. Jakob Nielsen [3] recommends that around three to four persons should perform the method as this provides the highest efficiency/cost ratio; with four evaluators it's estimated that 70% of the problems are discovered. Due to this fact, heuristic evaluation is often considered a very cost effective way to discover usability problems.

Heuristic evaluation can be performed in two different ways. One way is to have a single evaluator following an evaluation checklist and taking notes. The other is when a user experiments with the software under the supervision of an observer. The observer tries to identify potential problems during the testing session. The main difference between this method and the User interview method is that the user is allowed to ask questions. This is since, if the user's questions are answered, it will often grant an understanding of the user interface and save precious testing time.

The list of heuristics proposed by Jakob Nielsen [3] is:

- **Visibility of system status:** The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
- **Match between system and the real world:** The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.
- **User control and freedom:** Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
- **Consistency and standards:** Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
- **Error prevention:** Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.
- **Recognition rather than recall:** Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
- **Flexibility and efficiency of use:** Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
- **Aesthetic and minimalist design:** Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.
- **Help users recognize, diagnose, and recover from errors:** Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
- **Help and documentation:** Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

2.2 UNITY

The interface presented in this report has been implemented in Unity, a multiplatform game development environment. Unity is designed to allow for fast game development, sparing the user the work involved in creating a content pipeline, a scene graph and more. Its selling points include: content centered development; scripting in C# .NET or JavaScript; immediate mode graphical user interface; deployment to web player; and more.

2.2.1 CONTENT CENTERED DEVELOPMENT ENVIRONMENT

In a traditional game engine the development is code centered. That means that almost all assets are imported by creating import scripts which in turn load the content into the game engine. In addition, almost all objects are first defined in code and then associated with the appropriate assets.

Unity instead works in a content driven manner. This means that all assets are imported by dragging them into the unity editor. The assets can be organized and renamed if needed. The assets that are required in the game are added simply by dragging them from the editor asset window into the game assets window. Scripts are then added on top of the game assets in order to add functionality the game objects.

2.2.2 IMMEDIATE MODE USER INTERFACE

Programming graphical user interfaces (GUI) in an object oriented environment is usually done in retained mode. When working with Retained Mode GUIs the programmer sets up the interface in the initialization stage, creating all the different widgets used in it. This is sometimes done in some form of graphical editor. Each widget is given the data necessary to draw it, such as type (button, label, etc), position and what text or graphics to display on it. Each widget is also given an identifier of some form to use later. What happens in the actual update of the GUI is not known to the programmer since it is handled internally in the GUI library used. When a widget is used, e.g. a button is pressed, a callback is generated or a message is added to some message queue. The callback or message is handled by looking at the ID of the widget that generated it and then take the necessary steps. This can, for example, include getting some data held by the widget (e.g. the text in a text field) or setting an attribute of another widget. Since this happens in another part of the code - following the Model-View-Control pattern - the two parts must share the widget IDs between them. Also, since the widget classes used for the different widget types must contain all the data that needs to be available when the callback is generated, creating custom widgets and controls typically involves creating new classes.

In Immediate Mode the creation of the GUI follows most of the principals found in immediate mode graphics (as opposed to retained mode graphics); where each element is drawn in each frame by calling its draw method each update loop. Instead of assigning a position, contents and ID in the initialization stage, each widget is given a position and contents in the draw call that draws it. The same draw function also returns a value depending on whether the widget is used or not. For instance, the draw function for drawing a button returns true if the button is pressed. That way, the code handling the effects of the widget can be defined with the button itself. A simple example:

```
Update()
{
    if(Button(10, 10, "Press me"))
    {
        DoButtonAction();
    }
    ...
}
```

All data used by the widgets is stored at application level – e.g. the text shown in a text field is stored globally in the class - so adding new behaviors to a widget is comparatively fast and easy. This method of developing graphical user interfaces is particularly well suited for creating prototypes easily since everything needed to handle one widget is done in only one place in the code.

2.2.3 UNITY WEB PLAYER

Unity can deploy a created application in a web-player format which can be embedded on a web page. When a user opens the page he will be prompted to download and install the Unity Web Player (unless he has already done so). After installation, the application embedded on the page is executed. The web player runs at the client side and downloads all the information needed from a server which hosts the application. Since the web player runs like a script, it also means it has the same access rights as any other web script; it may not read and write to the local hard drive. Saving and loading can still be done since Unity contains support functions for creating and filling web forms. These can be used to invoke e.g. Perl and CGI scripts on the server. The Unity Web Player also has the capability to save primitive data variables in a Player Preferences object, which may contain only 1 MB data. This is often used to save user defined settings.

The web player can interface with the page it is hosted on, sending information to and from the application. The communication between the unity application and the host site is performed through Java script.

In deploying Unity projects to the web player, developers have the option of creating streaming applications. These run as normal applications, but only load a part of the data on startup. The rest of the data is streamed into the scene when certain events are triggered, defined by the developer (such as the player reaching a certain area).

2.3 3D NAVIGATION

Navigation in 3D environments has been researched by a great number of researchers and has spawned a large number of papers and reports. A distinction often made is between *Egocentric* navigation, where the user moves through space, and *Exocentric* navigation where a user wants to move around a given object. Examples of the two could be a flight simulator (egocentric) and a 3D modeling program, such as 3D Studio Max (exocentric). Ware and Osborne [4] examined three interaction paradigms for 3D navigation: *eyeball-in-hand*, *scene-in-hand*, and *flying*. Flying was found to be the best for egocentric navigation and scene-in-hand was found to be best for exocentric. Tan, Robertson and Czerwinski [5] divided navigation into three sub-tasks - exploration, to survey a scene or landscape; search, to find a specific object or route and travel to it; and inspection, to establish and maintain a particular view of an object. There have also been attempts at creating visual aids for helping the user navigate, such as landmarks (in this project called viewpoints), breadcrumbs, maps and compasses. Darken and Sibert [6] give an overview of several of these.

To alleviate the complexity of 3D navigation many researchers think control should be restricted for the user. The idea is that the creator of the 3D world knows best which areas or tours are of interest to the user and should therefore assist the user in reaching them, or perhaps even limit the user's movement to those areas, landmarks or tours. Several tests have shown that putting constrictions on the user’s movement helps complete navigational tasks more quickly and without the user getting disoriented. Several research articles present different forms and amounts of camera constraints and how they affect the user’s experience [7][8][9][10][11][12][13][14]. Some research [7][15] also discuss camera techniques where the user only has control in certain areas, surfaces or views and the movement between those views is handled automatically (without user control).

VR 3D navigation in everyday life is most often performed with two dimensional (2D) input devices, such as a mouse or a game-pad. A common navigation scheme for first-person computer games is to map horizontal movement - running forwards and backwards, strafing left and right - to keyboard buttons (typically WASD or arrow keys) and control the camera view rotation with the mouse. A common navigation scheme for 3D CAD programs and similar is to use the mouse for all forms of movement - often mapping the scroll wheel to zooming - and use modal keys (e.g. shift, ctrl and alt) to change between different actions.

Sundin and Fjeld [16] describe three transfer functions controlling how the device input is handled: *Position Control*, where the **position and orientation** of the controlled object are proportional to the position and orientation of the input device; *Rate Control*, where the translation and rotation **velocity** of the controlled object is proportional to the position and rotation of the input device; and *Acceleration Control*, where the translation and rotation **acceleration** of the controlled object is proportional to the position and rotation of the input device. Citing [17][18] *Acceleration Control* is not considered by Sundin and Fjeld for its difficulty to handle. In the first-person game input scheme mentioned above *Position Control* is most often used. This means that if the user moves the mouse slightly to the left, the view rotates to the left with an angle proportionate to the mouse movement, and then stops. Using *Rate Control*, the same movement will cause the view to start rotating slowly to the left and continue to do so. Moving the mouse further to the left will increase the speed of the rotation, while moving the mouse back again will decrease it. This method is sometimes used together with a so-called "dead zone" [19]. The dead zone is typically an area at the center of the screen in which, while the mouse cursor is inside, the rotation does not change.

3 RELATED WORK

The graphical user interface in this project is very specific and due to that fact related work has been hard to find. There exist a great number of guidelines concerning good ways to design simple user interfaces aimed at the novice user, but little was found in terms of articles assessing full interface solutions. For this project the interface design present in many computer games was turned to. These are often designed to be friendly to the first time user, and at the same time simple and easy to learn.

Although there is much research done on the subject of making 3D navigation more accessible to novice users very little of this research is applicable in its entirety for this project. One of the reasons is that the interface presented in this project should be easy to add into an already created 3D environment.

3.1 SPEED-COUPLED FLYING WITH ORBIT

Tan et al. [5] present a technique they call Speed-Coupled Flying with Orbit. In this technique the height and tilt of the camera is determined by the speed of the camera. The user moves the mouse forward/backward to move the camera forward/backward and right or left in order to turn the camera. The faster the mouse is moved, the higher the camera goes from the ground and the camera is also tilted more downwards at higher speeds. They also use a technique they call orbiting. When a user clicks and drags the cursor on an object, the camera is moved to a view over the object.

Tan et al. present two tests performed with these techniques. One test was performed with a 3D environment 300 by 300 meters containing 4 objects and the other was performed on an environment 500 by 500 meters containing 23 objects. Most objects were carnival-themed structures such as tents, roller coasters and rides. In both tests the technique was deemed very successful and preferred by most of the test subjects.

Speed-Coupled Flying is an interesting idea for this project, but it does not fit entirely. The technique presented in the paper seems to rely on the 3D world not being too densely filled with objects. In the test scenarios presented the densest world only has 23 objects, some of small size. The concern is therefore that this technique would not work well in an environment where the user can easily collide with objects, such as a city. The second problem is that the orbiting mode assumes there is a well defined way of finding a view of an object. This view could be created by the world creator, or be calculated automatically, but there is not always a good way of doing this for a specific object. The technique also does not allow the user to control the pitch of the camera manually, which can be useful in the kind of application used by Dynagraph.

3.2 AUTHORED VIEWPOINTS

In [20] Chittaro and Burigat and Abásolo and Della [15] systems using authored viewpoints are presented. Maganalles [15] is a tool which allows the user to navigate by selecting a viewpoint from a list of available viewpoints based on where the user is at the moment. The camera is then moved to that viewpoint through an animated gliding to help the user understand where he is moving. The Maganalles tool also allows for connecting viewpoints to define virtual tours.

Although Maganalles addresses the problem of navigation for novice users in a very good way, it requires data not always available in a 3D world. For the tool to be useful the world must contain an adequate number of landmarks between which the user can navigate. In an environment such as a town district this could mean a great number of viewpoints, especially since the locations of interest may vary between different users.

Chittaro and Burigat [20] present a navigational aid showing the user the way to a designated landmark. A user can specify a landmark he wants to visit and a 3D arrow is displayed showing the direction and distance to the target. The arrow is updated dynamically so it changes as the user moves.

The aid presented by Chittaro and Burigat fits well with the desired outcome of the project, but does not solve the problem of navigation. It is a good technique for adding a way for users to get a better overview and understanding of the environment, but should be added on top of an existing navigation control scheme.

3.3 MULTISCALE NAVIGATION

McCrae et al. present a navigation technique called Multiscale Navigation [19]. The technique is meant to allow users good control when navigating in completely different scales, from earth orbit to street level. Left clicking an object or surface starts the camera moving toward it with a dynamically changing speed. Any subsequent left clicks will retarget the camera's flight. The speed is dependent on the closeness to surrounding objects, decreasing if the camera is moving through a dense area. The camera is also repelled by close objects, so the camera may not always travel in a straight line towards a target object. While the camera is flying the user can look around by moving the mouse. Right clicking while flying towards an object stops the camera movement. Right clicking while stationary performs the opposite behavior to the fly-towards initiated when left clicking. The camera is pushed away from the object and away from the earth's center. If the push-out is used when indoors, the camera moves to the nearest exit before pushing away. If the user clicks the left mouse button while pushing away the camera stops moving. The user can look around while stationary by holding the right mouse button. The rotation of the camera is performed with a rate control transfer function and utilizes a “deadzone” in the middle of the screen. There are also two additional techniques available; HoverCam and Framed zoom. HoverCam allows the user to rotate around an object and is activated by dragging the left mouse button. Framed zoom allows the user to zoom in by drawing a frame on the screen, to which the viewport is fitted, and is activated by holding shift and dragging with the left mouse button.

Multiscale navigation is seemingly a good technique for this project although there may be some problems with it. The technique is not specifically aimed at inexperienced users but is rather an attempt at finding an efficient way of using the same system to navigate through several different scales. The technique is based on environments with a strong connection to orbit-to-ground navigation, moving either towards the ground from orbit or the other way. While the technique is developed with street level "horizontal" flying in mind, it does not address some of these issues. For example there is no definition of what should happen if the user left clicks empty air. Finally, the technique is far from trivial to implement in its entirety and quickly produced prototypes will probably not do it justice.

4 REQUIREMENTS

The requirements are stated informally and are mainly a list of functionality expected to be available in the final prototype. The non-functional requirements listed below are important properties taken into account in the design of the prototype.

4.1 FUNCTIONAL REQUIREMENTS

1. The interface should support free 3D navigation.
2. Viewpoints: Support a way to view stored locations (bookmarks) in the scene and be able to jump to a specific location. Possibly even support user-added waypoints.
3. Tours: Support a way to view stored tours (movies) and be able to jump to a specific tour. Should also support normal movie playing actions such as play, pause, stop, rewind, fast forward and tracking.
4. Save screenshot: Support saving a screenshot of the current view. User should be able to download the saved screenshot.
5. Model alternatives: Support viewing several different versions of the same 3D object, e.g. if there are three different bridge designs, the user should be able to select and view a specific design.
6. Measure: Support distance measuring.
7. Setting time for sun light: Support changing the time of the day, thus adjusting shadows and sun position accordingly.

4.2 NON-FUNCTIONAL REQUIREMENTS

1. Maintainability: Should be easy to maintain.
2. Expandability: Should be easy to add new features.
3. Easy to add to existing project: Should not require much work on the 3D world to function with it.

5 METHOD

This chapter will explain, in more detail, the design methods used to reach the final result. The testing performed is explained in 5.1 Testing followed by the design. Although the navigation system and the graphical controls were designed using many of the same methods and tested together, they are based on different design rationales. The iterations and steps in the two design processes will therefore be presented separately in sections 5.2 Navigation and 5.3 Graphical User Interface.

The first step performed in the design of the interface was to establish a target user group. Through discussions with Dynagraph, the common denominator for the intended users was determined to be a low to middling proficiency with computers in general and a very low proficiency with 3D applications in particular. Test persons for the user testing were chosen in accordance with this.

Since Unity allows for quick interface development - especially graphical interfaces - prototypes have been implemented in code directly, bypassing the need for alternative prototyping methods such as Paper Prototyping. The design process has followed a general work flow.

1. An initial design of a GUI element or control is created, often by looking at similar functionality in other applications or by looking at known design patterns.
2. A simple prototype is created to further test the interaction.
3. The prototype is evaluated with Cognitive Walkthrough and the design is corrected or adjusted according with results. At this stage the prototype may also evolve.
4. Heuristic Evaluation is used to find design or implementation flaws and the prototype is further improved based on the findings.
5. The prototype is subjected to user testing followed by user interviews.
6. The design is re-evaluated and redesigned based on the user tests.
7. Steps 3 to 6 may be repeated until the user tests give satisfactory results.

5.1 TESTING

Three forms of testing and evaluation have been performed; Heuristic Evaluation, Cognitive Walkthrough and Task Oriented Interviews with Talk-Aloud.

Heuristic Evaluation has been used in two forms. Firstly, we, the interface designers, have used the list created by Jakob Nielsen (presented in 2.1.4) to look for usability issues throughout the design and implementation process. Secondly it has been used is during the user testing. After the tasks given in the task oriented interview have been completed the users have been asked to comment and have been given questions pertaining to the list mentioned above.

Cognitive walkthrough has been used as presented in 2.1.3 with little deviation. The resulting reports presenting possible usability issues is presented in Appendix C.

The number of people used for testing varied during the project. A total of 12 test persons, 4 male and 8 female, with different degrees of skill have participated in testing the different prototype versions as well as the original application. Not all testers have performed every test so some versions of different interface elements have received very limited testing. The tests have been performed according with the script found in Appendix A. To allow for users to start using the system fairly quickly rather than allow for maximum efficiency, much of the testing has been focused on finding out what makes novice users intimidated or uncomfortable. Users have not been timed, but rather their comfort in using the system has been observed. An overview compilation of the test results can be found in Appendix B.

Most tests have been performed using an example project from the Unity website. The demo consists of a tropical island with some mountains, rivers and coasts. The island has little in ways of human development; a few huts and bridges.

Later tests have been performed on an early version of one of Dynagraph's model scenes. It has a number of block shaped buildings as well as a neighborhood of more detailed structures and scenery.

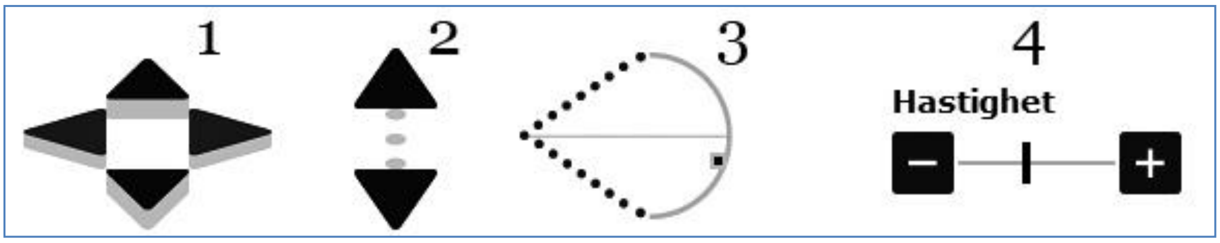
5.2 NAVIGATION

The navigation system design has differed from the general design flow in that a number of solutions have been presented to alleviate each perceived problem. To test the impact of a solution, the mechanism behind the solution has been placed in several different complete navigation systems. Each mechanism is presented separately together with the results and conclusions from testing it.

5.2.1 NAVIGATION: THE ORIGINAL APPLICATION

The navigation system in the previous application uses two different forms of navigation, one through control widgets in the tool bar and one through mouse movement. Both schemes are one-handed and only require the user to interact with the mouse. The widget system, seen in Figure 1, consists of four controls.

Figure 1: Navigation control widgets in the original application.



1. Horizontal movement: The arrows control forward and backward movement as well as rotation around the vertical axis (i.e. in the horizontal plane). Pressing and holding the forward and backward arrows will only move the camera in the horizontal plane and not upwards or downwards, even if the camera is angled upwards or downwards.
2. Elevation control: Pressing and holding the arrows will move the camera upwards or downwards. The control is not a slider, so there is no interaction available with the dotted line between the arrows. The controls move the camera along the global vertical axis, and are not affected by the pitch of the camera. Note that there are no widget controls for panning sideways.
3. Pitch control: Moving the green marker along the arc will cause the camera to pitch upwards or downwards.
4. Speed Control: This slider controls the speed of the cameras horizontal and vertical movement. It affects the movement speed both when controlling using the widgets and when controlling using the mouse controls presented below.

The second control system (hereafter referred to as the direct controls) entails holding one or two mouse buttons to activate different modes. Holding the left mouse button (LMB) and moving the mouse will rotate the camera around the current position. The rotation uses a rate control transfer function (see 2.3 3D Navigation).

Holding the LMB and right mouse button (RMB) while moving the mouse will cause the camera to move forwards or backwards while also rotating. For example, holding both buttons and moving the mouse forwards will make the camera move straight forwards while moving the mouse forwards to the right will make the camera move forwards while turning right. Both the forward movement and the rotation uses rate control transfer functions, i.e. the camera will move and turn faster the further from the origin the mouse is moved. The origin, in this case, is the position of the mouse when the buttons were initially pressed.

Holding the RMB only while moving the mouse will pan the camera. As with the other controls, the panning is done through a rate control transfer function.

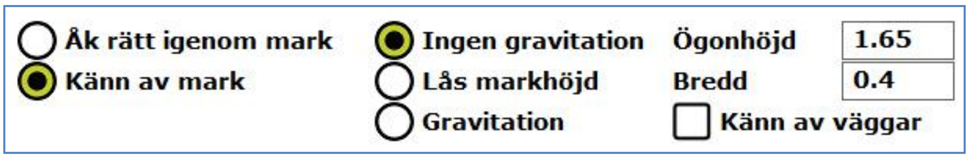

Åk rätt igenom mark
[✓ Känn av mark]
[✓ Ingen gravitation]
Lås markhöjd
Gravitation
Ögonhöjd 1.65
Bredd 0.4
☐ Känn av väggar

Figure 2: Navigation settings in the original application.

There are a number of navigation settings available to the user, most notably gravity and locked ground level.

While gravity is applied the camera will accelerate towards the ground when not grounded, simulating real world physics. Locked ground level allows the user to specify a height above the ground to view the world from, which for instance can be useful if one wants to view a street from the height of a car driver (approximately 1.1 m). This height is exactly kept without any physical simulation or similar.

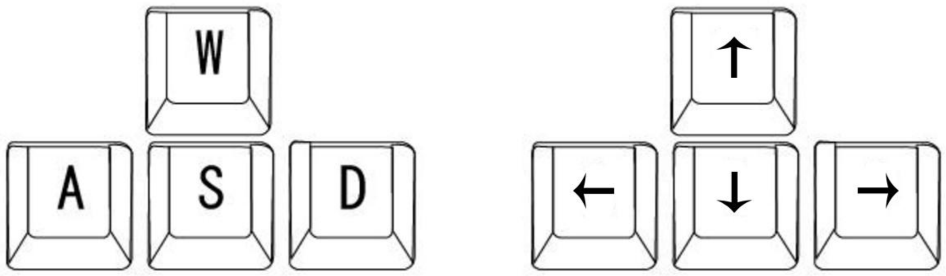
5.2.2 TESTING: THE ORIGINAL APPLICATION

In the initial navigation system users preferred not to use the control widgets, even when subtly encouraged to do so. This behavior persisted even when the user had problems using the direct controls. The main problem encountered in the direct controls was that the users did not understand that movement was separated between axes. Users repeatedly tried to navigate towards an object above or below their current position by rotating the view towards it and moving forwards. The users often did not even acknowledge that it had not worked as they intended, instead believing they had aimed poorly when rotating. Some users also exhibited problems keeping both mouse buttons pressed while navigating.

In the test sessions users were given a quick walkthrough of the navigation settings and later asked to navigate along street level to perform a set of tasks. Close to all users, even having been shown the settings, did not use the gravity or locked ground level settings. In most cases this was said to be because they had forgotten about the settings or did not know how to find them.

5.2.3 FIRST-PERSON WASD CONTROLS

The first-person WASD control scheme requires two hands, one to control forwards, backwards and sideways movement with the keyboard and one to control camera rotation with the mouse. Traditionally in implementations of this control scheme, forward and backward movements are mapped to W and S respectively while sideways movement (strafing) is mapped to A and S (hence the name WASD), with an alternative mapping to the arrow keys.



The mouse controls camera rotation and can be implemented with either a position control transfer function or a rate control transfer function. Three versions were implemented for testing, one using a position control transfer function and one using a rate control transfer function, both always rotating the screen when the mouse position warranted it. The third system also used a rate control transfer function, but required the user to press and hold a mouse button for the camera to rotate. Releasing the button stops all rotation.

5.2.4 TESTING: FIRST-PERSON WASD CONTROLS

When testing the WASD system the first problem that arose was that most test users found it hard to use both hands for navigating. Users who had no experience in computer games or similar navigated by first rotating the camera and then move the mouse hand to the keyboard to move the camera. When comparing between position control and rate control transfer functions, most users preferred rate control and exhibited less trouble in keeping their mouse in a comfortable working position. Comparing the always active rotation mechanism with the one requiring the user to keep a mouse button pressed, user expressed a lack of control in the former and greatly preferred the latter. This observation was supported by results from testing the Multiscale system.

5.2.5 MULTISCALE NAVIGATION

The Multiscale Navigation control scheme is based on a control scheme developed by Autodesk, detailed in [19]. The original versions intended use is for Multiscale applications such as Google Earth or Microsoft Virtual Earth, where the user moves from outer-planetary orbit to street level. Note that the version implemented for this project differs from the original in several ways.

A LMB click initiates a movement, making the camera fly towards the object or surface clicked. Subsequent LMB clicks retarget the flight. While flying, moving the mouse will rotate the camera, while keeping the same flight direction and clicking the RMB while flying will stop the camera’s movement. Clicking the RMB while stationary will initiate a push-out movement, moving away from the objet or surface clicked. A LMB click while pushing out will stop the push-out movement.

There are also two drag movements where the user clicks and holds a mouse button while moving the mouse. These are only available while stationary. A LMB drag will rotate the view letting the user look around. A RMB drag will initiate hover camera, which rotates the camera around the targeted object while keeping the rotation view on the object.

5.2.6 TESTING: MULTISCALE NAVIGATION

Testing and evaluation of the Multiscale control system showed that users felt a lack of control when the camera continued to move without the user actively interacting with it. This was observed, for example, when the user clicked a destination and the camera started moving. Several users forgot how to stop the camera movement (RMB click) and simply let go of the mouse, (some) becoming slightly panicked when that did not help. This observation together with the results when testing the WASD scheme led to the formulation of a navigation design principle for the project: navigation shall at all times be an active interaction and if the user stops interacting with the controls all navigation shall stop.

Users had very different opinions about the hover camera. Users with prior experience with or knowledge of similar mechanisms understood its control quickly and, in general, found it useful. Other users found it hard to understand and could not control it to their liking. Later tests showed that using a rate control transfer function gave user a sense of more control. The initial implementation of the hover camera followed the "scene in hand" mechanism [4] where the user rotates the scene rather than moving the camera. This proved problematic for most users to learn and a test was performed with an implementation where the camera rotation had been inverted; i.e. the camera moves the direction the user moves the mouse towards. This control change received a generally positive reaction from the users. One of the reasons for the success of this break from the "scene in hand" mechanism may be the large size of the scene. This theory is also supported by the test results found by Ware and Osborne [4].

5.2.7 CHASE CAMERA

The chase camera is based around the simple principle of following the point clicked by the mouse. Clicking and holding a mouse button makes the camera move towards the object the cursor is hovering over. The rotation of the camera is also turned towards the direction of the cursor at all times. The cursor can be moved during flight, thereby continually retargeting the movement. While this allows no way for the user to travel backwards, Tan et al. [5] observed that users rarely use backwards movement except for when they've overshot. Combining this scheme with one that allows for backward movement may therefore be adequate.

The speed of the camera is a weighted sum of the previous speed and the new speed (determined by the target distance). By adjusting the weight the designer can set how much the speed reacts to new target distances. If the previous speed has a large weight value the camera will react more slowly to new target distances, thereby making it accelerate slower but provide a smoother ride. Conversely, a small weight value will make the camera accelerate faster but the ride may be jumpy.

A chase camera can be implemented by shooting rays from the camera through the near clip plane, at the mouse cursors position and, upon hitting a target, setting the movement direction towards it while also rotating the view towards the target. Using this implementation the case where no surface or object is hit by the ray (the user could be flying into the sky) will also have to be taken into account.

The same interaction can also be achieved by using a rate control transfer function to rotate the camera according to the mouse position, setting the mouse origin at the center of the screen. This will rotate the view as long as the cursor is not positioned at the exact center of the screen, and rotate it faster the further from the center the cursor is moved. The movement direction is set to straight forward. This method is easier to implement, less computationally demanding and it already handles the case where the user is pointing into empty space.

Note that there is no inherent way of deciding the speed of the sideways movement for any of these implementations.

5.2.8 TESTING: CHASE CAMERA

The chase camera received positive reactions from most users some even stating it was a "fun" way of navigating. From observation we noted that the learning time was short and that users found themselves in few confusing situations while using the chase camera. One discernable problem was the lack of reverse control. Users often moved a bit further than they had intended and started looking for a way to move backwards.

5.2.9 MOVEMENT SPEED

For any control implementation only using the mouse there is a gap between the degree of freedom (DOF) in the input device compared with the desired DOF in the application navigation. This requires the implementation to restrict the freedom of control given to the user, e.g. not allowing the user to move in one direction while looking in another. A common limitation in 3D navigation implementations is the lack of speed control.

Movement speed in 3D applications, for instance the running speed of the player avatar in a first-person game, is often constant. When the application allows user movement at different scales, however, this may become somewhat inadequate. The speed appropriate for moving along a street sidewalk may very well feel excruciatingly slow if the user wants to fly over the rooftops to the other side of town. Three methods are proposed for setting the movement speed, of which the later two use automatic speed setting.

5.2.9.1 SCROLL WHEEL SPEED CONTROL

Using the scroll wheel to set movement speed is a method used in the Splinter Cell games. It allows the user control of the speed at all times without using extra input devices. When moving, the movement speed is increased by scrolling up, and decreased again when scrolling down. If a movement has stopped and then continues, the speed may either be reset - requiring the user to scroll up again to increase it - or stay at the previous set speed, depending on implementation. While resetting the speed means repeating the scrolling more often, it also means that an unproficient user can simply stop the movement if he/she feels it is going too fast, and be given back control to try the speed setting again. Using this method coupled with a control method requiring the user to press and hold mouse buttons may be problematic, since scrolling while holding a mouse button can be hard, especially if it is the left mouse button.

5.2.9.2 CAMERA HEIGHT SPEED CONTROL

Setting the movement speed based on the height of the camera is based on the assumption that the user wants to traverse longer distances and is less interested in details the higher up he gets. This is often true in VR environments of nature, for instance, where the user is only interested in details while close to the ground and is more interested in moving far when high up. The case of VR environments of urban areas does not fit as well with this generalization.

There are two ways of defining camera height: The distance between the camera and the surface below the camera; or the distance between the camera and a defined "zero-level" (e.g. sea level). Using the former can cause problems if there are many fluctuations in the landscape height - for instance in a city with great differences between the height of the building rooftops and street level - which will cause the speed to constantly change. Using the latter may introduce problems if the details in an environment are spread between different height levels - for instance in a city where the user should be able to move around a rooftop with the same precision as when on the street.

5.2.9.3 TARGET DISTANCE SPEED CONTROL

A solution to the problem of mapping inadequate input to the desired navigation controls may be to use modal keys (often ctrl, shift or alt) to allow for more input combinations. A major disadvantage with modal keys, however, is that they may be distracting for the user, especially if the user is not proficient with similar applications. In [5] it is suggested finding ways of automatically setting modes based on the user’s intentions. For this project, this idea has been explored by using automatic moding for setting movement speed.

The idea is to set the movement speed based on the distance to what the user is heading towards. This method allows for the desired speed control without the flaws of the previous two methods, except that there is natural way of handling sideways movement. The method is easy to implement by shooting a ray forward from the camera and scaling the speed with the distance to the surface or object hit. The case where no hit is detected is handled by setting a maximum speed.

5.2.9.4 TESTING: SPEED CONTROL

Controlling the movement speed with the scroll wheel was tested with the WASD and Multiscale control schemes. In both cases users found the added complexity hard to get used to and preferred to navigate without using it.

The speed changing based on camera height was, in most tests, not noticed by the user. When the mechanism was turned off, however, users exclaimed irritation at the slow speed, flying over the scenery. One test person noted the (too) high speed when flying over a mountain path, which he wanted to examine in more detail.

As with the height-based speed control, the distance based speed control was not noticed by the users until the mechanism was turned off. In both cases (height- and distance-based) the users responded more positively to the general navigation than in previous tests when the dynamic speed control had not been implemented.

5.2.10 LIMITING USER MOVEMENT

Unproficient users often end up in confusing camera rotations or positions when navigating in a 3D environment. There are two such situations commonly encountered: the user moves too close to a vertical surface making the entire screen show the same color or texture, or even cause clipping; or the user rotates the camera to one of the extreme rotations looking straight up or straight down. The latter is confusing, both because any subsequent rotation around the vertical axis will cause the view to spin and because, when looking down, the user cannot see a body, thereby breaking the metaphor of controlling a human being and seeing through his eyes. Some research on the subject suggests restricting the user's navigation in different amounts (see 2.3 3D Navigation).

The restrictions tested in this project are fairly small, limiting the allowed pitch angle of the camera and the closest possible distance to surrounding objects. The pitch restriction is implemented by allowing the user to rotate the camera 60° up and 60° down from the horizontal axis.

The restriction on how close the user can come to objects in the environment is implemented by setting a disproportionately big size on the bounding volume used for detecting collision on the camera object.

5.2.11 TESTING: LIMITING USER MOVEMENT

The limits imposed in the tests resulted in the least experienced users getting confused or disoriented less often. It had little impact on the more experienced users. However, no users acted or remarked on the restrictions in a negative manner.

5.3 GRAPHICAL USER INTERFACE

When designing a usable graphical user interface (GUI) there are several key principles to follow, two of which are affordance and feedback. Affordance is a property or quality of an object which tells the user they can interact with it. Common affordances are buttons being given a 3D look, to give the impression they can be pressed down, or buttons being highlighted when hovered. Feedback is used to notify the user that something has changed, for instance assuring the user that an interaction is complete and has had consequences. These two principals are followed throughout the design process even if not explicitly mentioned.

The GUI is divided into several tools. Some of the tools share similarities in implementation and have been prototyped in parallel, to create a set of consistent patterns for the complete interface. For many of the interface parts, however, the development has started later or it has been suspended for periods, resulting in test sessions testing different iterations of the different tools together. Here they are presented separately, to give a structured view of the implementation and testing.

5.3.1 MAIN INTERFACE

The main interface consists of the GUI elements visible to the user at all times. In this method description we also include the menus used to open the different tools under this heading.

5.3.1.1 MAIN INTERFACE: THE ORIGINAL APPLICATION

In the original application the lower part of the screen consists of a toolbar where all controls are displayed. The toolbar has a number of tabs which the user can click to display the corresponding controls in the toolbar panel.

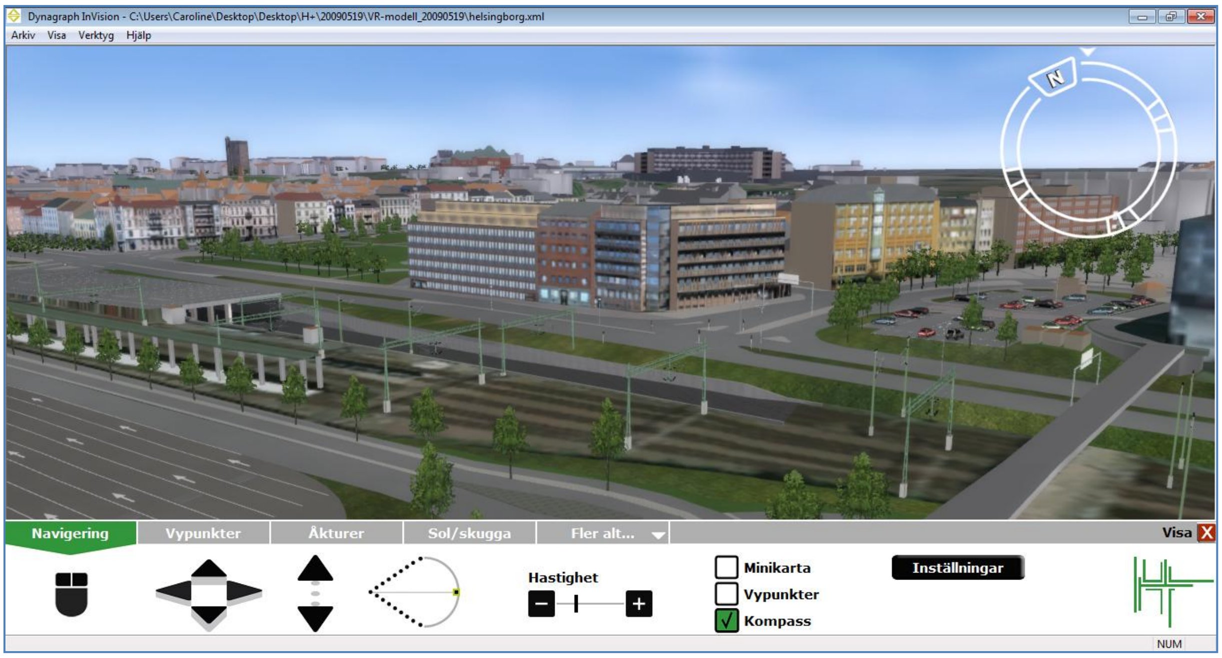


Figure 3: The interface in the original application

The toolbar can be closed and reopened by the user to allow for a bigger screen. The control widgets for navigation - displayed when starting the application - are presented in 5.2.1 and the controls for each other tool is presented in that tools section.

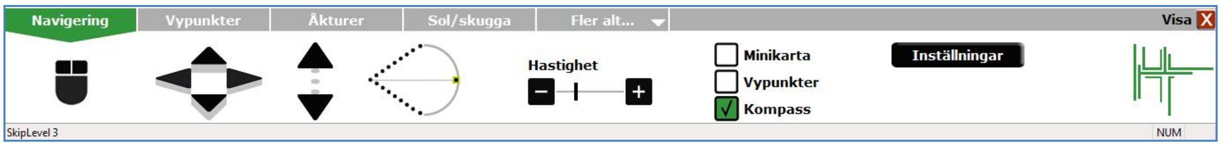


Figure 4: Toolbar in the original application

The testing of the original toolbar showed that, even though the toolbar is large, users often overlooked available controls and settings. Users spent most of the time in the 3D scene not using or taking note of most of the controls in the toolbar, but when they wanted to use a control they often found it hard to locate. Even though users were asked to close the toolbar, when completing the task they opened it again, in spite of the fact that they did not intend to use any of the controls on it. There was also confusion about whether the navigation tab needed to be open in order to use the mouse navigation controls in the scene.

5.3.1.2 MAIN INTERFACE: PROTOTYPE 1

In the first prototype of the main interface, an attempt was made to make the more important navigational functions easily available to the user, without using too much screen space at all times. This was done by putting a slide-in menu on the right side of the screen. This menu has two tabs, "Viewpoints" and "Tours", giving easy access to the two corresponding lists. Moving the mouse cursor to the left of the screen highlights the menu button and clicking it will make the tabbed list menu slide into the screen. Different sizes and forms were tested for the button opening the side menu.

The interface has a tool menu, opened by clicking a menu button in the lower left corner of the screen, in keeping with familiar precedents such as the Windows start menu. This menu gives access to all tools not available in the side menu, such as saving screenshots, model alternatives etc.

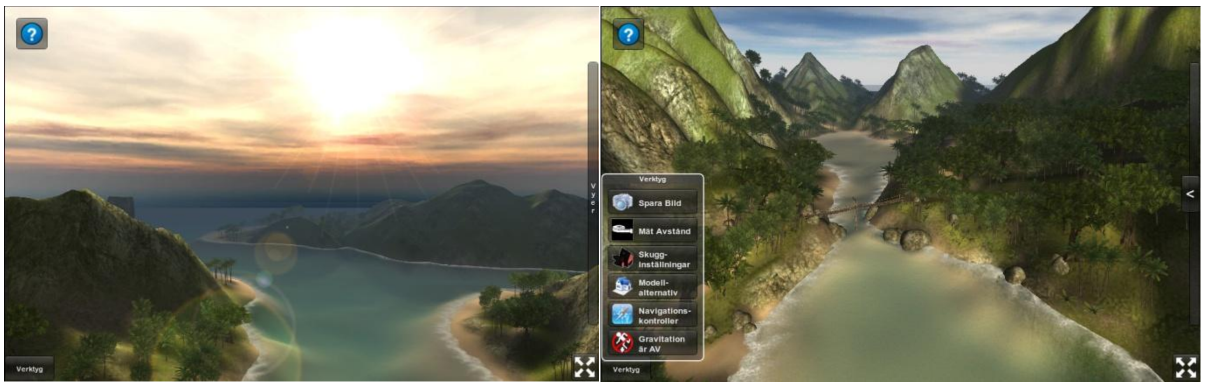
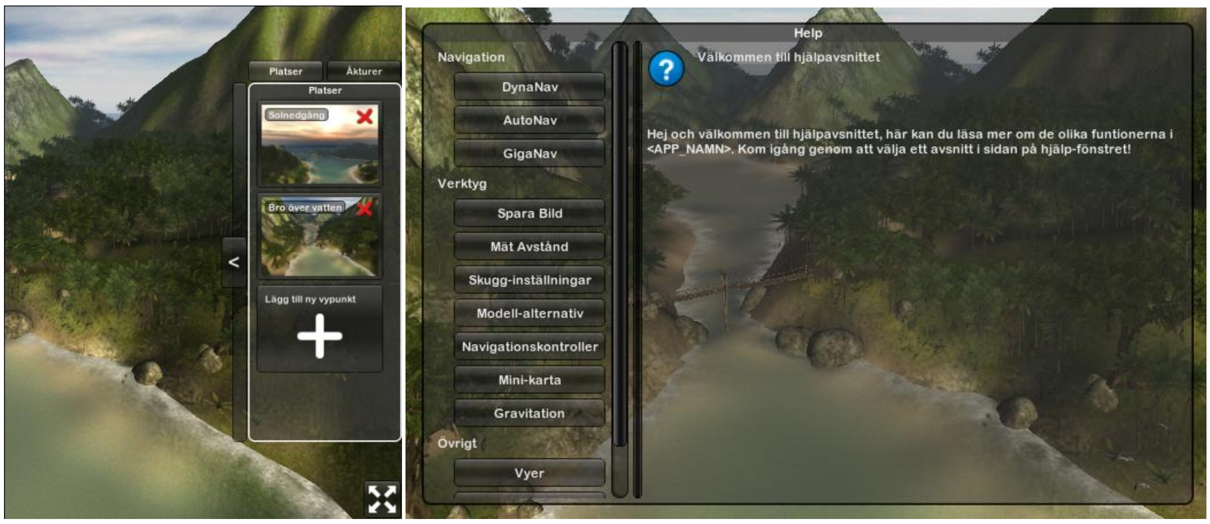


Figure 5: Two versions of an early main interface prototype. The two versions use different side menu buttons.

There are also two quick buttons available to the user, one for changing the screen mode between full screen and normal mode and one for displaying a help window. The screen mode quick button changes appearance when clicked to show that its function has been reversed. The last button in the tool menu is a gravity toggle. As with the screen mode button, it changes appearance when clicked.



Hej och välkommen till hjälpavsnittet, här kan du läsa mer om de olika funtionerna i <APP_NAMN>. Kom igång genom att välja ett avsnitt i sidan på hjälp-fönstret!

Figure 6: On the left: Open side menu. On the right: Help menu

The testing performed on the first prototype showed a general behavior on which many of the design decisions in the final prototype of the application are justified. The behavior was that, the more experience a user has with computers in general and similar applications in particular, the more likely he is to simply try clicking until the desired result is reached. Users not comfortable enough at the computer screen are anxious or nervous to "break" the application or end up in an irreversible situation.

The above mentioned behavior made most users automatically look in the tool menu regardless of which task we asked them to perform. The familiar pattern of a start menu allowed them to search for the correct tool in the menu without fear of what might happen. When asked to jump to a specific viewpoint users would automatically assume they would find the correct button in the tool menu and, when they did not, were prone to give up or ask for help. In some cases users noted the viewpoints/tours button on the side of the screen, even going as far as to hover it with the cursor (causing it to highlight), but were still afraid to click it uncertain of what would happen.

The quick buttons for setting the screen mode and gravity were often misunderstood, both because users did not understand the icons and because users were not expecting these features to be accessed in the main interface directly, instead looking for them in the tool menu. The help button and corresponding menu was removed from future prototypes since similar functionality was being developed by Dynagraph in another part of the project.

5.3.1.3 MAIN INTERFACE: PROTOTYPE 2

The second main interface prototype is based on the principle of using familiar patterns as much as possible. All tools are available through the tool menu which is a part of a toolbar, somewhat similar to the taskbar in Windows. This toolbar also has a toggle, of a very traditional make, for toggling gravity as well as a quick button for opening the Viewpoint window. Again, the choice of making viewpoints more readily available is because they are considered an important navigational tool.

Later this toolbar was extended to allow for further quick button functionality, should the user want it. The toolbar can be extended to hold an additional quick button and the target tool of the quick buttons can be set by the user (see 5.3.8 Settings Window). The default setting, however, is to show one quick button connected to the Viewpoint window.

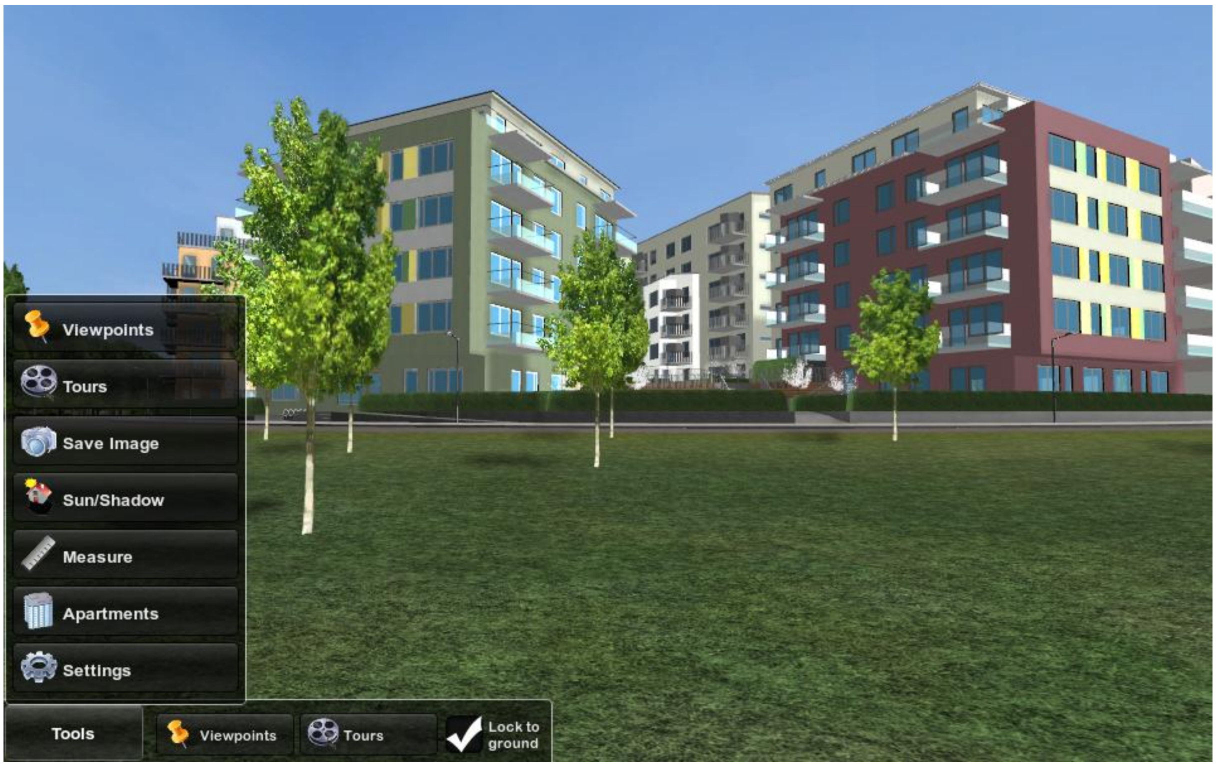


Figure 7: Final main interface with the tool menu open.

The second main interface prototype was received with positive reactions. Users easily found the tools they were looking for, and used the quick buttons when opening tools, if available. Users also understood the purpose of the gravity toggle or were not afraid of testing it if they did not.

5.3.2 VIEWPOINTS

The viewpoint tool is a tool which should allow the user to jump to specific views created by the environment creator as well as create his own views which he can later return to.

5.3.2.1 VIEWPOINTS: ORIGINAL APPLICATION

The version of viewpoint controls available in the original application consists of a drop down list with a text entry for each viewpoint. Clicking a viewpoint with the LMB in the list will move and rotate the camera to the corresponding view. The text field below the list displays the name of the currently selected viewpoint, which is the viewpoint last clicked.

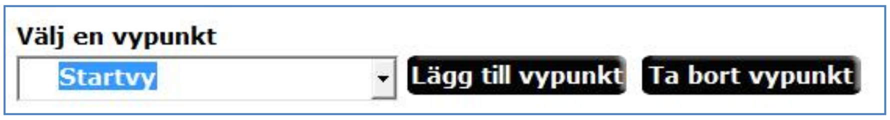


Figure 8: Viewpoint controls in the original application.

There are two buttons available to the user: *Add new viewpoint*; and *Delete viewpoint*. Using the former opens a dialogue window for giving a name to the new viewpoint. When the user clicks OK in the dialogue, the new viewpoint is added to the list, but the user is not notified and has to open the list to make sure. To delete a viewpoint the user first selects it in the list and then clicks the Delete viewpoint button. The user is asked to confirm the deletion of the viewpoint and clicking OK will remove the viewpoint from the list. The text field below the list will, however, still display the same name, i.e. the name of the viewpoint deleted.

Clicking a viewpoint in the list with the RMB will give the user access to several functions through a drop-down menu. These include renaming the viewpoint, giving a viewpoint a new view, importing and exporting viewpoints plus a few more. The intended use of the rest of the functions is unknown, since their purpose did not become apparent during evaluation.

A setting is available to display thumbnail versions of the viewpoints in the main interface. These are displayed on the side of the screen as small pictures without visible names and are limited to a certain number. The thumbnail viewpoints can be clicked in order to jump to them, but the right click menu is not available through them.

The main problem found when testing the viewpoint controls was the lack of feedback. Users were often unsure of whether their actions had had any effects. In a few cases the user, upon being asked to delete a viewpoint, deleted two viewpoints since the text field still, after deletion, displayed the name of the viewpoint the user wanted to delete.

Users did not attempt to use the right click menu, for lack of understanding about the functions available through it. Also, the thumbnails were not used since users found it hard to distinguish between different viewpoints with only the small picture. Some users noted, however, that they liked the overview of the viewpoints the thumbnails gave them.

5.3.2.2 VIEWPOINTS: PROTOTYPE 1

The first prototype developed for viewpoint control is a one column list which is part of side menu presented in 5.3.1 Main Interface. Each viewpoint is represented by a thumbnail and a name tag. Hovering over the viewpoint button will highlight it as well as display a red X in the top right corner of the button. Clicking the button will move and rotate the camera to the view, while clicking the red X will delete the viewpoint. When deleting, the user is presented with a dialogue to confirm deletion.

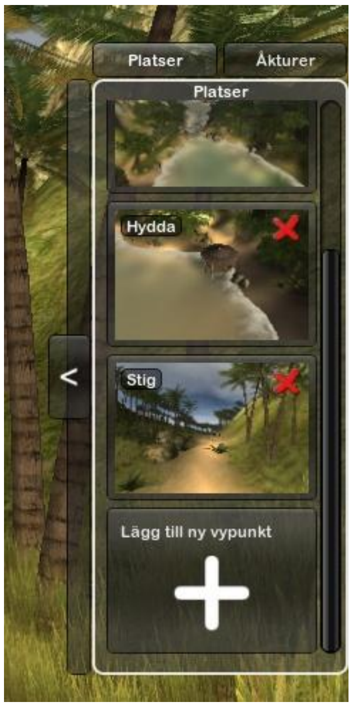


Figure 9: Viewpoint side menu, here with the brush stroke *X* icon used in the final version.

At the top of the list is a button for adding new viewpoints. When adding a viewpoint, the user is presented with a dialogue for giving a name to the viewpoint.

During testing users found the controls for adding viewpoints and jumping to viewpoints easy to understand. The *X* icon for deleting viewpoints, however, received mixed reactions. Some users understood its purpose immediately, other associated the *X* with closing a window and thought clicking it would close the side menu. Others still did not see the *X* saying it was hard to distinguish from the background. Several users tried right-clicking when asked to delete a viewpoint

5.3.2.3 VIEWPOINTS: PROTOTYPE 2

The second prototype of the viewpoint controls consisted of two separate versions. The reason for this was that we identified two very different user groups for this tool. One user group (hereafter called tourists) needs simple controls for utilizing a strictly limited number of interesting views. The other group (hereafter called brokers) uses the tool to keep track of housing objects such as apartments in an environment. A broker may have as many as two viewpoints for every apartment object in the environment, which in worst case can mean around 100 viewpoints.

To handle these two very different user groups, two different versions of the viewpoint system were proposed. The tourist version consists of a window with a list of viewpoints, each viewpoint represented by a thumbnail button with a name tag. At the top of the window is a button for creating a new viewpoint, which opens a dialogue asking the user to supply a name. Clicking a viewpoint button with the LMB will make the camera "jump" to that view. Each viewpoint button also has a red brush stroke X in the top right corner, which appears when the button is hovered. Clicking it will open the deletion confirmation dialogue used in the previous version. Clicking a viewpoint button with the RMB will open a drop-down menu allowing the user to delete or rename the viewpoint, both of which are done through a dialogue window.

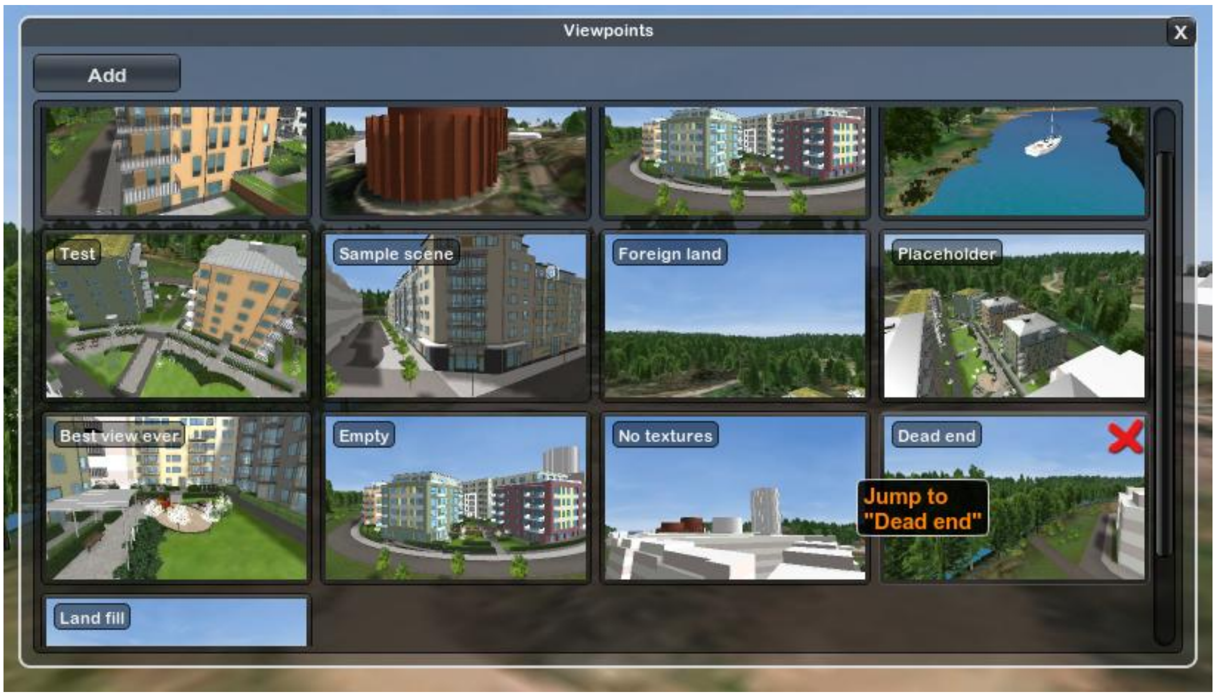


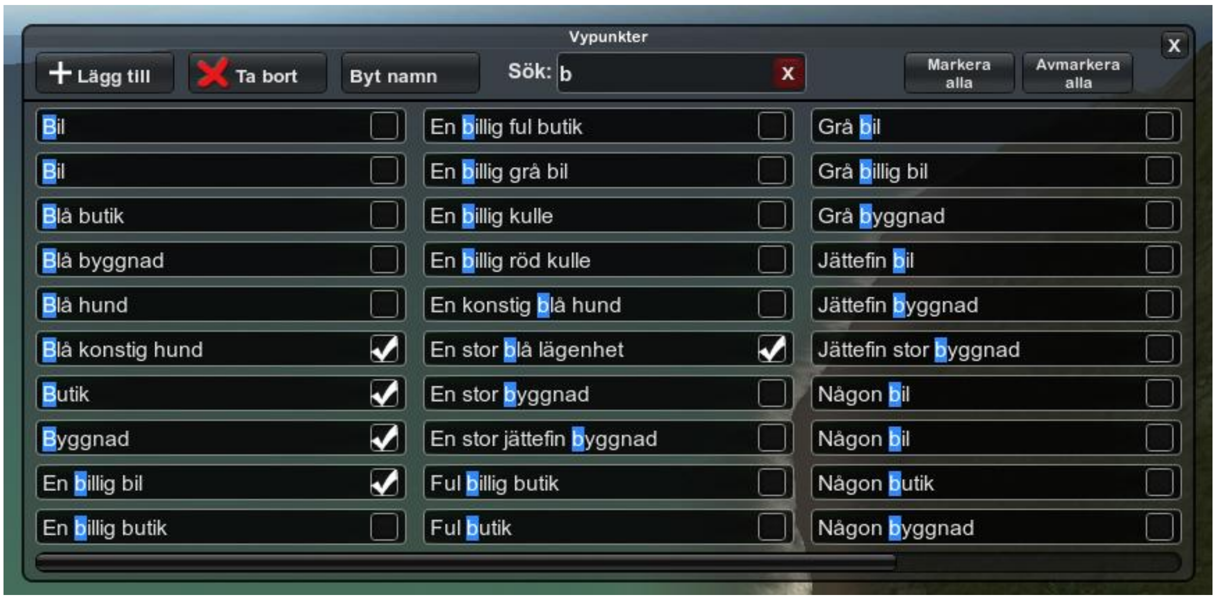
Figure 10: Viewpoint window in the final prototype.

The broker version of the viewpoint controls differ significantly from previously tested versions. The window consists of a columned list with a sideways scrolling scrollbar. Each viewpoint in the list is represented by a text button displaying the name of the viewpoint. Each button also has a toggle field for selecting several buttons. At the top of the window several controls are available. There are buttons for creating new viewpoints, renaming viewpoints and deleting viewpoints. There is also a search field as well as buttons for marking or unmarking viewpoint buttons.

Creating new viewpoints works the same as in the tourist version. Viewpoints can be deleted or renamed by marking them (setting their toggle) and using the corresponding button. When the user deletes a viewpoint a dialogue asks the user to confirm the name of the viewpoint to delete. If several viewpoints are selected prior to deleting the confirmation asks the user to confirm the number of viewpoints to delete rather than the name. When renaming, the user is asked to supply a new name for the viewpoint. If the user has selected several viewpoints when renaming, he is notified of this and warned that a number will be added after the supplied name to separate the viewpoints from one another. Deleting and renaming a viewpoint is also available through a right-click menu.

The user can use the search field to quickly find specific viewpoints. Entering a search string will automatically update the list to only show viewpoints which names contains the search string. If viewpoints in the list have been marked they will be displayed regardless of whether they match the given search string, but if they don't they will be moved to the end of the list. The search string will be highlighted in the names which match it.

To the far right of the tool bar at the top of the broker viewpoint window are buttons for marking or unmarking all viewpoint buttons. Clicking Mark all will toggle all viewpoints on, and clicking Unmark all will toggle them off.



Bil		En billig ful butik		Grå bil	
Bil		En billig grå bil		Grå billig bil	
Blå butik		En billig kulle		Grå byggnad	
Blå byggnad		En billig röd kulle		Jättefin bil	
Blå hund		En konstig blå hund		Jättefin byggnad	
Blå konstig hund	✓	En stor blå lägenhet	✓	Jättefin stor byggnad	
Butik	✓	En stor byggnad		Någon bil	
Byggnad	✓	En stor jättefin byggnad		Någon bil	
En billig bil	✓	Ful billig butik		Någon butik	
En billig butik		Ful butik		Någon byggnad	

Figure 11: Viewpoint window with added functionality for handling large number of viewpoints.

Testing of the tourist version of the viewpoint system showed positive reactions. Users understood the controls and the feedback given to them at all times. The new style of the red *Delete* X – now drawn with brush strokes – on the viewpoint buttons alleviated the confusion previously observed with the symbol.

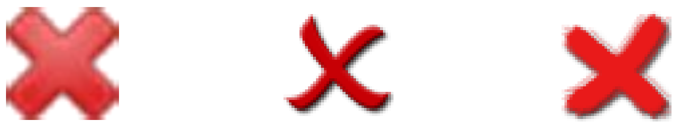


Figure 12: Left: The initial icon used for viewpoint deletion, Middle: A second tested icon, Right: The final deletion icon.

Upon evaluating the broker version of the system it was decided that it would be replaced by a separate broker tool not presented in this report.

5.3.3 TOURS

The tour control system should allow a user to select a tour to take and then control how the tour is played with play, pause and stop controls and similar. The system does not need to allow the users to create their own tours. When a tour is played the camera is moved along a prerecorded path and rotates according with prerecorded rotations. Tours can be used to give a user an overview of an area or environment.

5.3.3.1 TOURS: ORIGINAL APPLICATION

The tour control system available in the original application consists of a drop-down list for picking which tour to play, standard movie player controls such as play and stop as well as settings for the play-back.

The play controls are not initially visible, save for the tracking slider. When the user has selected a tour from the list a play button appears below the tracking slider.

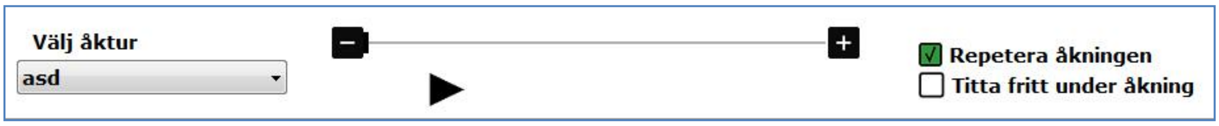


Figure 13: Tour controls in the original application

Clicking play starts the tour with a marker moving along the slider, showing the progress. When the tour has started the play button is made invisible again and pause and stop buttons appear instead. The slider allows the user to fast-forward or rewind by using the plus and minus buttons at the edges of the slider, or to track by clicking the desired point on the slider.

There are two play-back options available to the user: *Repeat tour*; and *Free rotation when playing*. Setting the former will make the tour begin over again when reaching the end. Setting the latter will make the user capable of rotating the camera view while the tour is playing.

The only notable problem found when testing the tour controls was that several users did not see or understand the play button. We established part of the reason for this being the lack of affordances; the button does not change in any way when hovered nor does it appear to be raised from the background, being only a flat symbol. When users found the play button, however, they also inferred the purpose of the pause and stop symbols.

Through evaluation and discussion with Dynagraph it was also decided that users should not have the ability to rotate the camera freely when in a tour, since this removes control from the environment designer and defeats part of the purpose of a tour; to give a directed overview of an environment.

5.3.3.2 TOURS: PROTOTYPE 1

The first prototype implemented of the tour controls was part of the side menu presented in 5.3.1 Main Interface. Each tour is represented by a thumbnail button showing the starting view of the tour as well as a name tag. The intention was for the thumbnail to show a small preview of the tour when the button was hovered, but this was not implemented in this prototype version. Hovering the button makes play controls visible over the button. When no tour is playing and the user hovers over a button a play symbol is displayed. Hovering over the play symbol specifically will highlight it and clicking it will start the tour. If the user clicks the tour button without clicking the play symbol the camera is moved to the starting position of the tour. Once a tour is playing its button will display two symbols at all times; pause and stop. Clicking *pause* will stop the tour and replace the pause symbol with a play symbol. Clicking the stop symbol will stop the tour, if it is playing, and move the camera to the starting position. It will also replace the two symbols on the button with a single play symbol.

When testing the first prototype of the tour controls, users found the first tour controls hard to understand. Several users found the controls hard to find and even after finding the play button they did not notice pause and stop buttons.

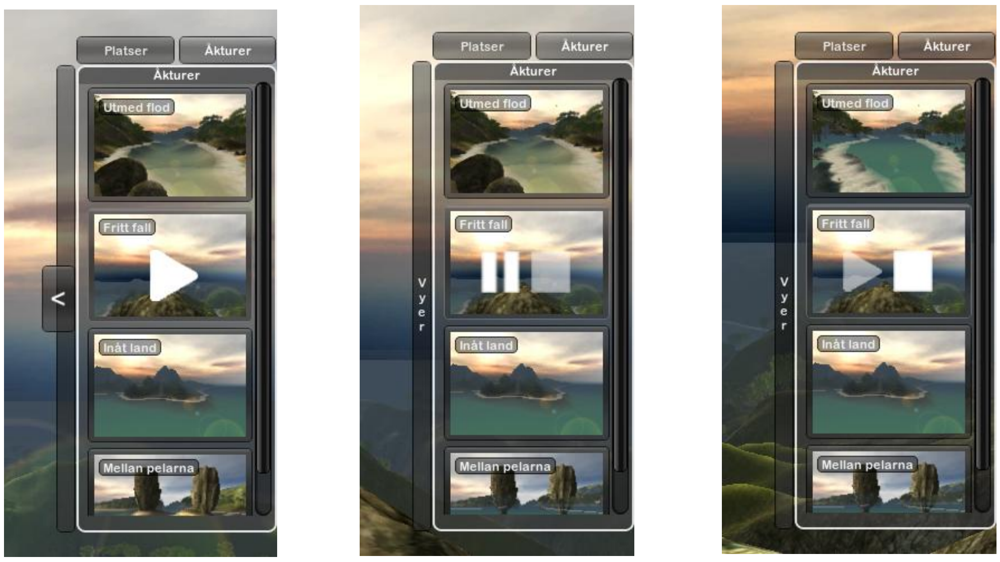


Figure 14: Left: A tour button with the play symbol highlighted, Middle: A tour button with the pause and stop button visible, as shown when the tour has been started, Right: A tour button with the play and stop buttons, as shown when paused.

5.3.3.3 TOURS: PROTOTYPE 2

The second prototype of the tour controls consists of two windows. When opening the tool the user is presented with a selection window, where each tour is represented by a thumbnail and a name tag. When a tour button is hovered the thumbnail displays a small preview of the tour. Clicking a tour will close the selection window and open a control window on the bottom of the screen. The play control window contains standard play controls as well as a button for choosing a new tour. Clicking the "Choose new tour" button will open the selection window again while keeping the control window open as well.

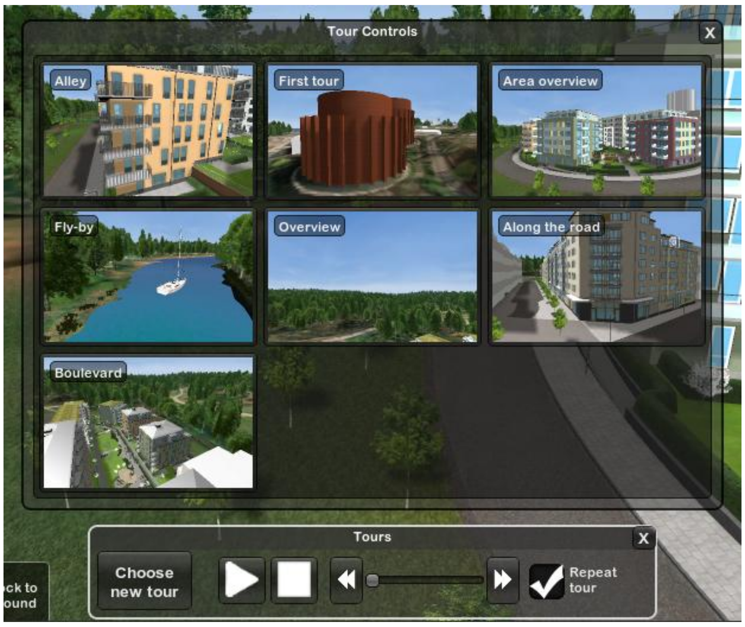


Figure 15: Tour controls in the final prototype.

The play controls available in the control window are play/pause, stop, rewind and fast-forward. The user can select a point in the tour by clicking the tracking slider. There also is a toggle for repeating (looping) the tour. If the user starts a tour, pauses it and then moves from his current position, clicking play will "snap" the camera back to the last position in the tour and the tour will continue.

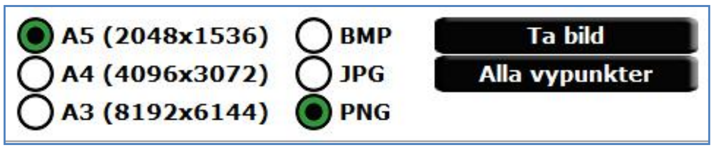
No noticeable problems were found while testing the second prototype of the tour controls. Users understood the function of the controls and had no problems completing the different tasks.

5.3.4 SAVING SCREENSHOTS

The controls for saving screenshots should allow the user to save the current view as an image file on the hard drive.

5.3.4.1 SAVING SCREENSHOTS: ORIGINAL APPLICATION

The screenshot saving controls in the original application gives the user the ability to choose size and format. There are three sizes available presented in standard paper size and in pixels. The formats available are BMP, JPEG, and PNG. When the user is content with the setting he can choose to either save the current view or save all viewpoints as images. The files are saved in a folder inside the application running directory.



[✓ A5 (2048x1536)] A4 (4096x3072) A3 (8192x6144)

BMP JPG [✓ PNG]

Figure 16: Controls for saving screenshot in the original application.

During testing many users did not understand the different choices available and either used the settings already set or used settings they had been told to use by Dynagraph personnel.

5.3.4.2 SAVING SCREENSHOTS: PROTOTYPE 1

The first prototype for the screenshot saving controls is a further test to see if the choices available to the user can be presented in a way that would make users understand them. The controls consist of a window with the same choices as the initial application, but with the size instead presented as small, medium and large. The user is also asked to supply the application with a directory for saving the image in, this in a test to see if users understand the concept of directory paths or similar.

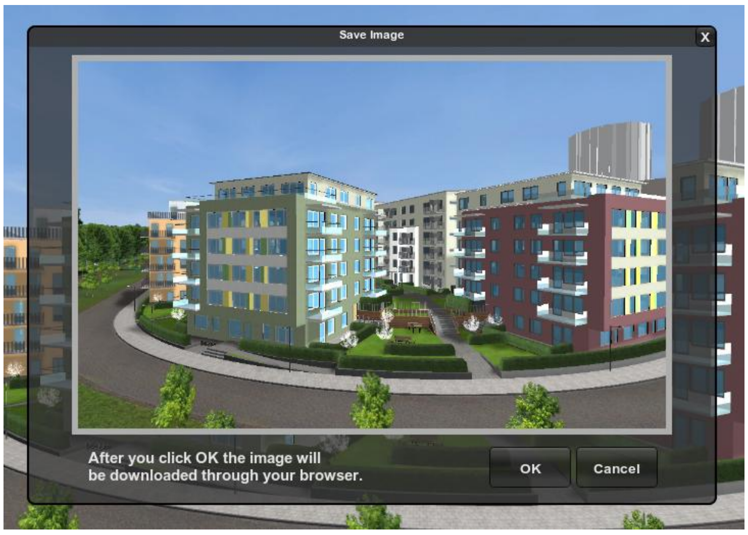
The testing of the first prototype gave the same results as with the initial application, in some cases confusing the user even more.

After discussions with Dynagraph it was decided that users should not get to chose size or format, but instead be given the image in a standard format and size decided by Dynagraph. This was decided to make the function more available to people not familiar with image formats and sizes. Since the finished application would run in a browser, the method of saving the screenshot also needed to be rethought, since Unity Web Player has no access to the local hard drive.

5.3.4.3 SAVING SCREENSHOTS: PROTOTYPE 2

The second prototype of the screenshot saving controls consists of a single large window. When the tool is activated a preview of the screenshot being saved is displayed inside a frame at the center of the window. The user is given text instructions saying that the image will be downloaded through the browser when the user clicks OK. The actual creation and download initialization is handled through Perl, CGI and JavaScript.

In the testing of the second prototype, no negative issues arose from users' interactions.



After you click OK the image will be downloaded through your browser.

Figure 17: Save screenshot controls in the final prototype.

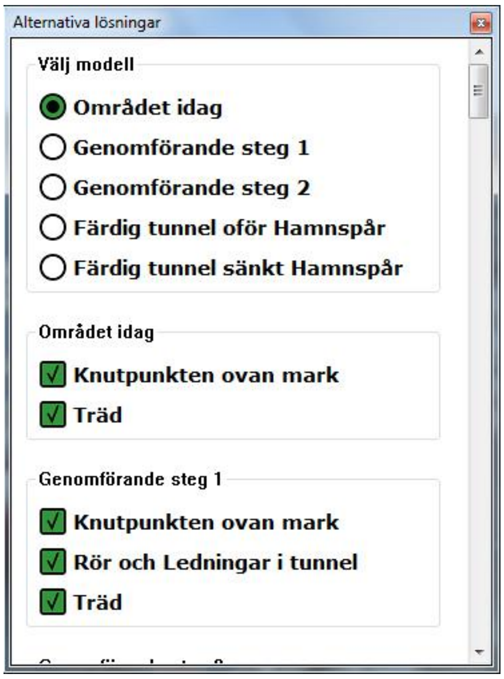
5.3.5 MODEL ALTERNATIVES

The model alternatives tool should display alternatives available for the currently viewed 3D environment. The environment creators could, for instance, have created several different versions of a park area and want to give the user the ability to change between the different versions.

5.3.5.1 MODEL ALTERNATIVES: ORIGINAL APPLICATION

The model alternative tool in the original application is named "Alternate solutions" and consists of a single window with a scrollable list of options. There are two types of options: two-state options, which are either on or off; and option list options, which lets the user pick one solution to be active out of a list of options. The two option types are realized as simple toggles and radio button lists respectively.

Although the controls for model alternatives were considered easy to understand in the initial version, users were confused by the lack of visible feedback. Activating or deactivating has no visible effect if the camera is not showing the affected area. Since the users were not familiar with the test environments, the names of the different options meant nothing to them, so they did not know where to look for the changes when changing the settings. This led to the users not really knowing if anything had happened at all.


Alternativa lösningar

Välj modell
- [✓ Området idag]
- Genomförande steg 1
- Genomförande steg 2
- Färdig tunnel oför Hamnspår
- Färdig tunnel sänkt Hamnspår

Området idag
- ✓ Knutpunkten ovan mark
- ✓ Träd

Genomförande steg 1
- ✓ Knutpunkten ovan mark
- ✓ Rör och Ledningar i tunnel
- ✓ Träd

Figure 18: Controls for alternate solutions in the original application

5.3.5.2 MODEL ALTERNATIVES: PROTOTYPE 1

The first prototype of the model alternatives window was only intended to test the general structure of the controls and only consists of the two-state type alternatives. The alternatives are represented by panels in a scrollable list, each panel having an image, a heading, a description and an on/off switch.

The heading and description are provided to give more details to the user about what the alternative changes or affects. The image illustrates a view showing the area affected by the alternative or a viewpoint representing the alternative. Clicking the image will move and rotate the camera to the corresponding view, giving users a way of locating a position from where they can see the changes.

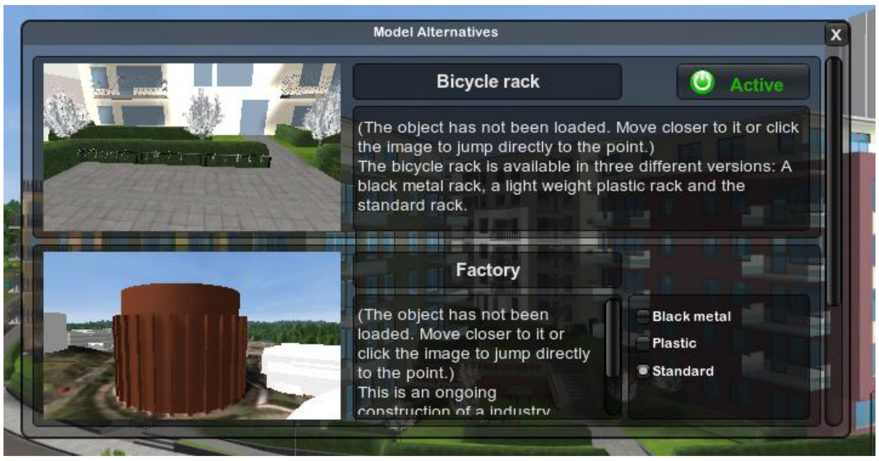
Next to the switch is a label showing the current state of the alternative. The label and button changes appearance when the alternative is toggled. The button text says "Activate" in green text and the label says "Inactive" in red text when the alternative is inactivate, and when the alternative is active the button says "Deactivate in red text and the label says "Active" in green text.

As with the initial version users found it hard to know whether or not anything had changed when they changed an alternative state. Few users tried clicking the images. Some users were also confused by the button and label changing color and text. One user stated that it felt as if the text was changing places when she clicked the button and it did not help her determine the state of the alternative.

5.3.5.3 MODEL ALTERNATIVES: PROTOTYPE 2

The second prototype of the model alternatives window is closely based on the first version. The image displayed on each panel is replaced with a real-time camera, so any change made in the scene is shown directly in the image. The label displaying the state of the alternative has been removed and the button instead show the state as "Active" in green text when the alternative is active and "Inactive" in red text when the alternative is inactive.

There is an additional type of panel to represent multi-choice alternatives. This consists of the same elements as the two-state type, but instead of an on/off switch there is a radio button list. If the user clicks one of the options the others are deselected and the scene is updated accordingly.



(The object has not been loaded. Move closer to it or click the image to jump directly to the point.)
The bicycle rack is available in three different versions: A black metal rack, a light weight plastic rack and the standard rack.

(The object has not been loaded. Move closer to it or click the image to jump directly to the point.)
This is an ongoing construction of a industry

Figure 19: Controls for model alternatives in the final prototype.

To allow for streaming the content in an environment, there is an additional case that has to be considered. The content affected by a model alternative setting may not be loaded at the time the user wants to change it. If this is the case, the description of the alterative is exchanged by a standard message telling the user that the object has not been loaded. The user is prompted to move closer to the area of interest or click the image to jump to it directly. When doing either of these, the user will trigger the streaming of the object and be able to manipulate the alternative.

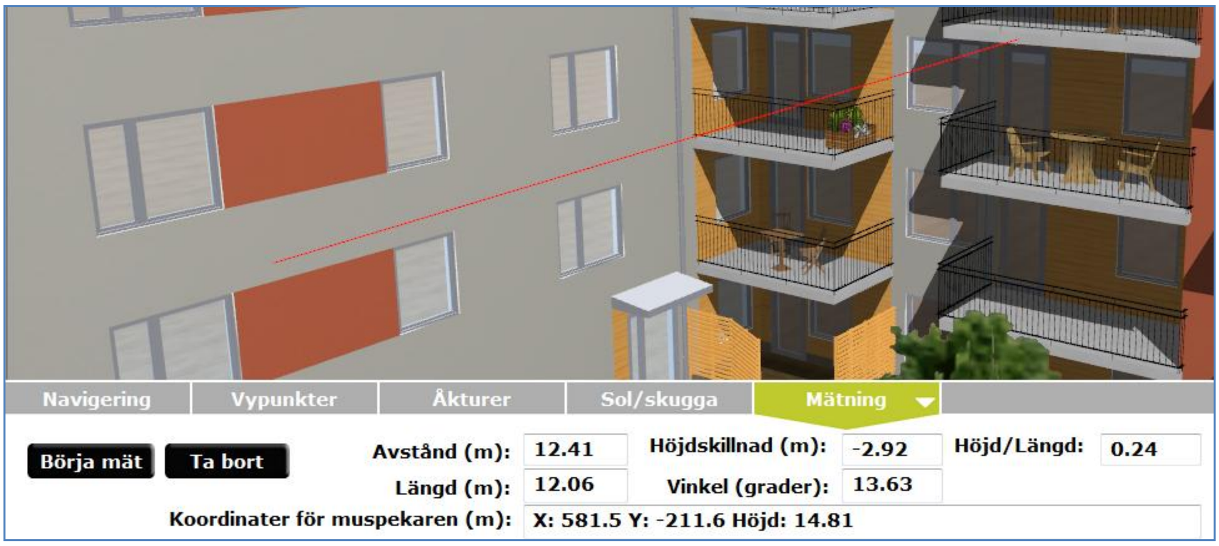
There was no formal testing performed on the second prototype of the Model Alternatives Window. The evaluation with cognitive walkthrough showed two possible usability problems. The first is that the user may not know that he can jump to the view of the model alternative by clicking the image, since there is no affordance suggesting as much. The second is that the text on the state setting button may be confusing to some users since it shows the current state of the alternative instead of what the button actually does. This is similar to a power button on a TV. From the evaluation results together with the fact that Dynagraph was satisfied with the solution further design of this element was not deemed necessary.

5.3.6 MEASURING DISTANCES

The tool for measuring distances should allow the user to select two positions and present any relevant data for the distance between those positions.

5.3.6.1 MEASURING DISTANCES: ORIGINAL APPLICATION

The measuring tool in the original application consists of two buttons – *Start measurement* and *Remove measurement* – as well as a number of text fields for presenting the results. The user starts a measurement by clicking the start button, selecting a starting position and selecting an ending position. When the starting position has been selected a red line is drawn from that position to the surface point hovered over by the mouse cursor. When the user clicks again, the line's end is attached to the clicked surface. Clicking the remove button will clear the drawn line and reset the presented values.



Avstånd (m): 12.41 Höjdskillnad (m): -2.92 Höjd/Längd: 0.24
Längd (m): 12.06 Vinkel (grader): 13.63
Koordinater för muspekaren (m): X: 581.5 Y: -211.6 Höjd: 14.81

Figure 20: Measurement controls in the original application.

The values presented are distance, length, height, angle of ascent, length/height ratio and the current world coordinates of the mouse cursor.

When testing the initial measurement tool several users overlooked the *start measurement* button. They clicked on different point in the scene and were confused when nothing happened. There was also some confusion as to what some of the measurement results meant. Particularly the angle of ascent and height/length ratio was puzzling to most users. None of the users saw the point of displaying the mouse coordinates.

5.3.6.2 MEASURING DISTANCES: PROTOTYPE 1

The first prototype of the measurement tool was based on the concept of the red line used in the initial version. When starting the tool the user is immediately put in measuring mode and can select the starting position. When the starting position has been selected a red line is drawn from it to the surface point hovered over by the mouse cursor. A label will also be drawn on this line displaying the current length of the line. When the user clicks again, selecting an ending position, the red line is stationary and two additional lines are added. A green line displays the height difference of the measurement and a blue line displays the horizontal length of it. Both these lines are drawn with corresponding labels displaying the results in numbers.

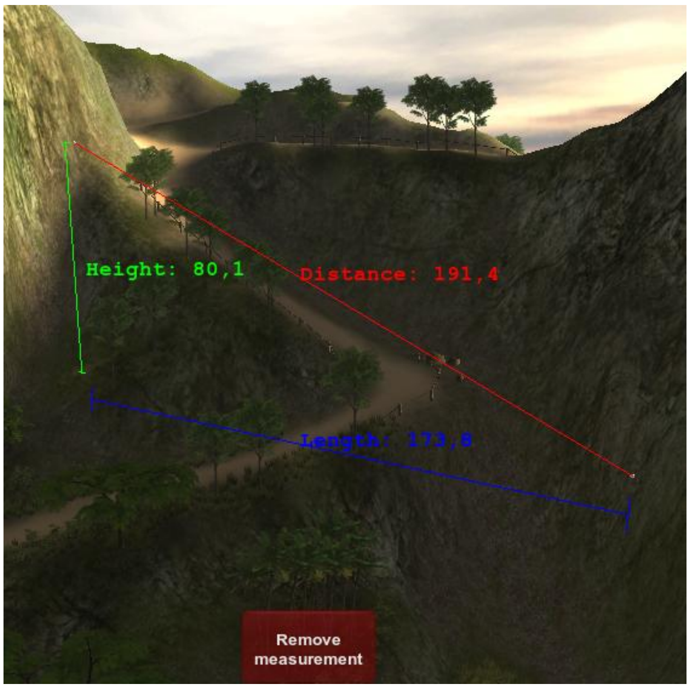


Figure 21: The first prototype of a measurement tool.

There is a red button displayed at the bottom of the screen. This button allows the user to clear a measurement. To start a new measurement the user starts the tool again from the menu.

When testing the measurement tool prototype almost all users found the lines drawn to be hard to understand. If a user measured a distance between two points approximately equally far away from the camera but at different sides of the screen, the resulting figure was understood in general. If the user made a measurement between a point close to the camera and another point far away, however, the use was confused as to how the lines were drawn. Although the lines were drawn as if in 3D they had no volume and were drawn “over” the terrain, which was confusing for the user.

An annoyance noted by the users was that, if the user was not satisfied with a measurement he would have to start the tool again from the menu.

Several users had problems hitting the intended point when starting or ending a measurement. They tried to hit the corner of a building’s rooftop or similar, and were not notified that they had missed. In such cases the users were confused by the fact that the measurement had not started/ended.

Through testing and evaluation together with Dynagraph, it was decided that the measurement tool should be simplified only giving the user a single distance, a not presenting height difference etc.

5.3.6.3 MEASURING DISTANCES: PROTOTYPE 2

The second measurement tool prototype used a single red line to present the measurement to the user. As with the previous version, the measurement is started when the tool is. To decrease excise, however, a button is present at all time in the bottom of the screen, allowing the user to start a new measurement at any time. A finished measurement is cleared when a new measurement is started or when the tool is closed.

When the tool has been started a sphere is attached to the mouse cursor. Clicking on a surface will "anchor" the sphere on it, giving the user a better view of where the line is attached. As before the line is drawn between the starting point and the cursor while moving the cursor around. A second sphere is attached to the cursor at this point and clicking a surface will "anchor" that sphere to it, thereby completing the measurement. Apart from showing the user more clearly where the measurement starts and ends, the size difference of the spheres gives the user a better understanding of the distance between the two points and the camera. The size of the spheres are set when the first anchor is placed, based on the distance between it and the camera. If the first anchor is set far away, the size will be bigger, to allow the user to see it properly. This means that if the second anchor is placed close to the camera, it will appear to be quite large. To make sure it is not in the way, it will get gradually more transparent the closer to the camera it is placed.

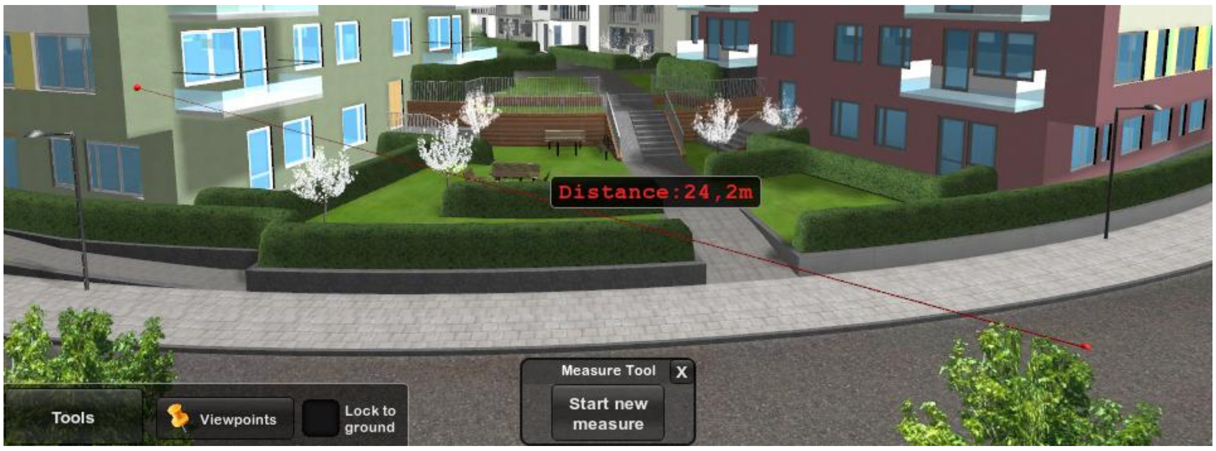


Figure 23: Measurement tool in the final prototype.

The result of the measurement is presented in a single label attached to the middle of the red line. It only states the distance between the two points.

The placement of the starting and ending position allows for slight deviations. If the user clicks in empty air, but is close to a surface point, that surface point is used instead. If the user is not close to any surfaces he is notified by an error message appearing at the top of the screen. The error message states that the measurement must begin and end on a physical surface and stays for a number of seconds or until the user successfully places the anchor.

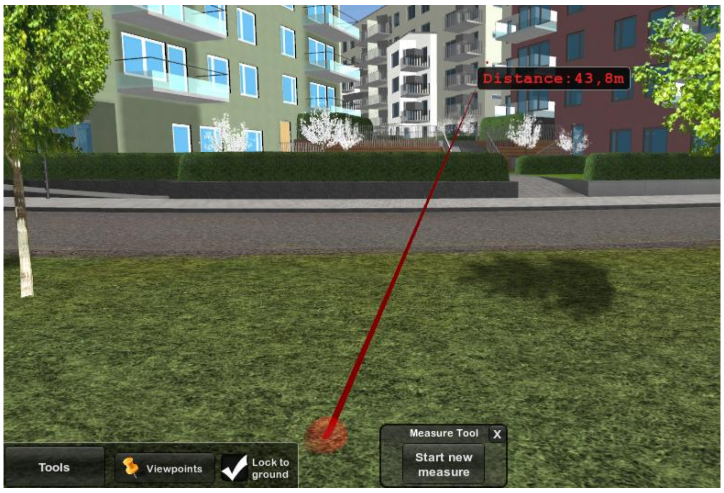


Figure 22: Screenshot showing the transparent effect on the anchor when it gets close to the camera.

The only objection observed during the testing of the second measuring tool prototype was that it was not possible to show several measurements simultaneously, which might be beneficial to some users.

5.3.7 SUN SETTINGS

The sun settings tool is used to decide the date and time of day to simulate in the VR environment. The date and time affects the sun's "position" in the sky, which in turn affects shadows in the environment.

5.3.7.1 SUN SETTINGS: ORIGINAL APPLICATION

The sun settings controls in the original application consist of two text fields for setting the date and time, a toggle for setting whether or not to show shadows and a slider for setting the shadow opacity.

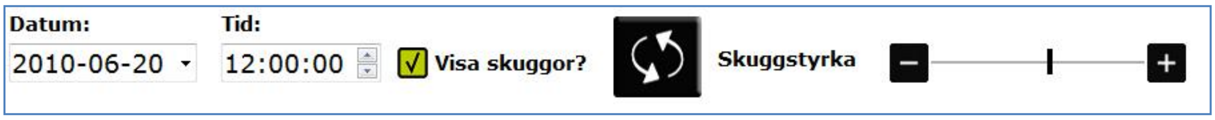


Figure 24: Sun setting controls in the original application.

The date field can be changed by writing a new date in it or by using the calendar opened by the drop-down button to the right of the field. Similarly the time field can be changed by writing a time or by increasing or decreasing the time using the buttons to the right of the field.

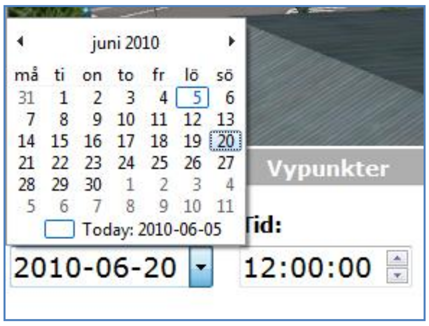


Figure 25: Date and time setting controls.

Making changes to the date or time or toggling the shadows makes an update button appear. This must be clicked in order for the changes to apply in the scene.

When testing the sun settings controls, almost all users overlooked the update button. Several users toggled the shadows on and off repeatedly, confused then nothing appeared to happen. The main reason for this was believed to be lack of or insufficient affordances.

5.3.7.2 SUN SETTINGS: PROTOTYPE

The prototype version of the sun settings controls consists of two windows. When starting the tool the user is presented with a control window at the bottom of the screen. This window has a button displaying the currently selected date and two text fields displaying hours and minutes as well as buttons for increasing and decreasing the hour. The time fields can be changed by typing a new time or by using the buttons.

Clicking the date button will open the calendar. This lets the user select a year, a month and a day. The calendar follows a general pattern used for calendars.

Since shadows are implemented with a different technique in Unity than the one used in the previous application, the shadows can be updated dynamically and there is no need for an update button, nor is there an option for disabling shadows in the environment.

The testing of the sun settings prototype showed no noticeable problems. Users were comfortable with the familiar patterns and immediately understood the controls.

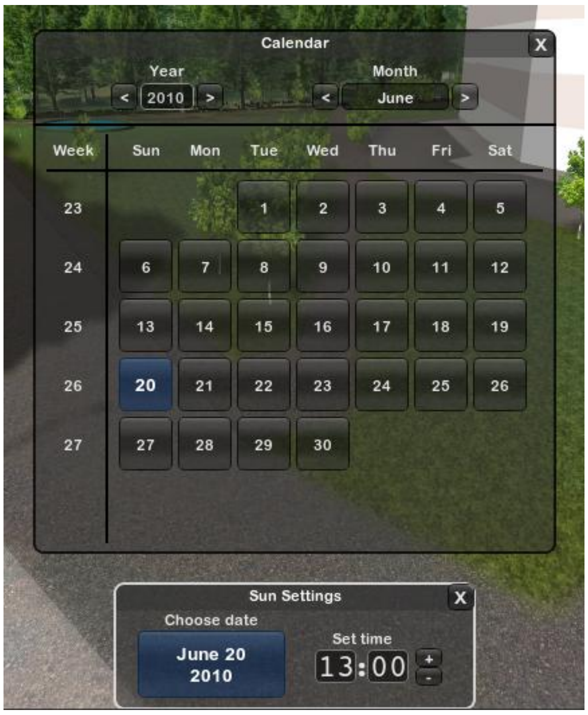
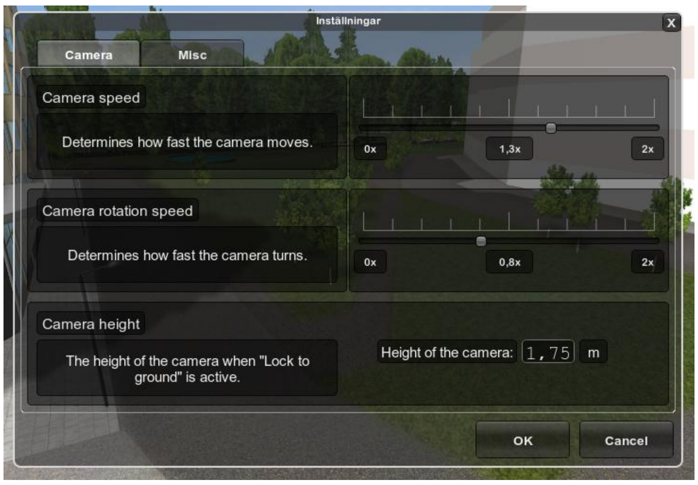


Figure 26: Calendar and time setting controls in the final prototype.

5.3.8 SETTINGS WINDOW

To accommodate the settings used by several other GUI tools in the interface as well as the settings used by the navigation, a settings window was added to the application. There is no similar window or tool in the original application, so no comparison is possible.

The settings window consists of two tabbed panes. The first, labeled “Camera” lets the user set the camera movement speed, the camera rotation speed and the camera height. The first two are available as sliders allowing the user to move the slider handle to a desired setting. The third is a text field allowing the user to specify how far above the ground the camera should be positioned when gravity is active. This setting lets the user see the environment from the eyes of a child, a very tall man or perhaps a car driver.



Camera speed

Determines how fast the camera moves.

0x 1,3x 2x

Camera rotation speed

Determines how fast the camera turns.

0x 0,8x 2x

Camera height

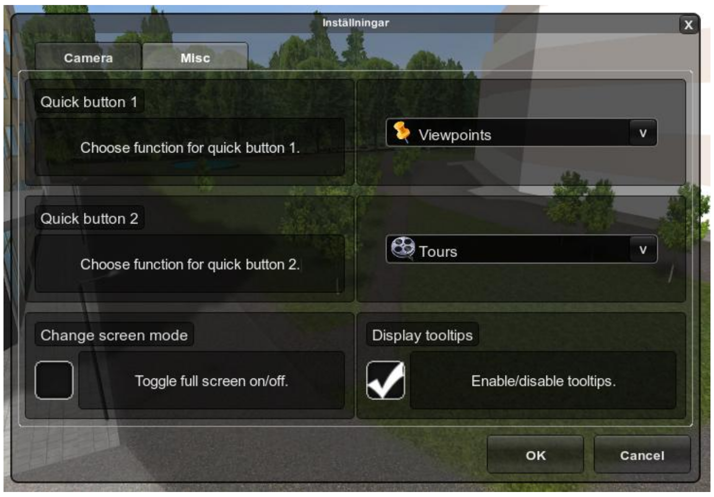
The height of the camera when "Lock to ground" is active.

Height of the camera: 1,75 m

Figure 27: Camera settings pane in the settings window.

The second pane is labeled “Misc” and has four setting controls. The first two are settings for the quick buttons in the main interface toolbar. The user can chose which tools should be available through quick buttons, or set the button to “none” thereby removing it from the toolbar. The last two settings are toggles for full screen and tooltips.

Settings changed are applied when the user clicks OK.



Quick button 1

Choose function for quick button 1.

Viewpoints

Quick button 2

Choose function for quick button 2.

Tours

Change screen mode

Toggle full screen on/off.

Display tooltips

✓ Enable/disable tooltips.

Figure 28: Miscellaneous settings pane in the settings window.

When testing the settings window users where confused by the quick button setting menus, since setting a buttons tool had no discernable affects (the user having not clicked OK). This confusion often cleared when the user tried clicking OK, but feedback to the user notifying it about the setting taking effect only after clicking OK would be preferable.

6 RESULT

In this section the final interface is presented, divided into navigation and GUI. Large parts of it are presented through reference to the corresponding version presented in the Method section. Each method or GUI design was chosen based on testing and how well it worked with the entire system.

6.1 NAVIGATION

The navigation control scheme is a mix of several of the controls presented, based on evaluation and test results. Visible to the user at all times is a toggle which the user can use to activate and deactivate gravity. There is also an option for the user to set the camera height when gravity is active. This setting is found in the Settings Window presented 5.3.8 Settings Window.

Holding the LMB lets the user rotate the camera, using a rate control transfer function. The user can combine this with moving by using the WASD or arrow keys. An alternative way of moving the camera is by holding the RMB which initiates the chase camera. The navigation speed is constant while gravity is active. When gravity is not active forward and backward speed is set based on the distance to the object or surface in the center of the screen, while strafing speed is set based on the height of the camera (measuring form the surface below the camera).

When testing the Chase Camera 8 out of 10 testers preferred it to any other they had tested. The remaining two both had previous experience with 3D games and both preferred the WASD controls. For more detailed description of the test results, see Appendix B.

Holding the space bar while clicking and holding the LMB will initiate the hover camera. It uses a rate control transfer function and the input axes are inverted relative to the *scene-in-hand* control scheme. When the mode is initiated a blue sphere appears at the focus of the camera, acting as an anchor for the user to know which point the camera is moving around.

Holding both mouse buttons allows the user to pan. This mode was not thoroughly tested but was added for the users who want it. It does not interfere with the other navigation techniques.

6.2 GRAPHICAL USER INTERFACE

The heads-up display of the GUI is implemented as described in the second prototype version with an addition. Clicking on a quick button once opens the corresponding tool, as before. If the user clicks the quick button again, however, while the tool is open, the tool is closed. This is not in line with the quick buttons of the Windows task bar, but is similar with an open window in Windows. To notify the user of this behavior an animation is used when opening or closing a window through a quick button. The animation gives the appearance that the window slides out of the quick button. The same animation, but reversed is used when closing the window by clicking on the quick button.

The viewpoint window and tour selection window are resized according with the number of elements in them. A low number of elements will make each element larger and the window adapt to a low number of elements. There is, however, an upper bound on the number of elements and adding more than that will add a scroll bar to the windows.

Other than the exceptions presented above the tools in the GUI are implemented as presented for the last prototype version of each tool (see 5 Method).

7 DISCUSSION

The development has been very diverse with several different tools being implemented and tested. This coupled with a limited time has imposed a behavior on the design in which we move on to the next tool implementation as soon as an acceptably low number of concerns remain in the current testing.

The testing performed on the prototypes developed in this project showed that the result were the best of what was tested, but there are many things we have not considered, for different reasons. As stated previously another part of the project, designing and testing the content available on the web site housing the application, was being developed simultaneously with this one. We therefore did not consider some of the focus of that development, mainly user tutorials and navigation aids.

The introduction of user tutorials or other forms of assisted user learning opens up new possibilities, especially when designing the navigation controls. If the user can be helped through the initial confusion when trying to use a system, he may, before long, find solution "A" preferable to solution "B" even though solution *B* is easier to understand at first glance. This directly correlates to the fact that expert users of a software application rarely prefer the same controls as a beginner.

The Speed-coupled Flying with Orbit [5] technique was not implemented for the reasons stated in 3.1 Speed-Coupled Flying with Orbit. The idea of coupling the camera speed with the camera height was examined, though in another form than the one found in the article. Ultimately the distance-to-target speed control was chosen instead since it performed better in a city landscape when evaluating. A more extensive test of the possibilities of this mechanism could definitely be beneficial though.

Apart from maps, which were developed in the other project, there are a great number of navigational aids useful for this type of application. We chose not to look into this for the simple reason that we had a limited amount of time and needed to limit our problem area. For future work, it would be interesting to test an implementation of Magallanes [15] incorporated with the controls scheme presented in this project to allow the user a quick way of orienting itself in a big environment.

As expected when researching the Multiscale Navigation [19] the simplified prototype was not well received with testers. Out of three testers all preferred other navigation techniques. The prototype did not set the navigation speed according to surrounding objects, nor did it use path finding and collision repelling to avoid obstacles. It should be noted that the technique received only limited testing since the tests seemed fairly conclusive. The article detailing the original implementation does not include user testing results and so it is possible even the full implementation is not particularly friendly to the novice user.

One of the most challenging parts of the project was to refrain from designing interfaces more suited to our own tastes. Since the project was about designing with simplicity and accessibility as first priority, more "interesting" interface ideas were scrapped for the sake of what worked with the users; well known patterns.

Another difficulty was in the selection of appropriate design methods. There are methods that, in hindsight, would have suited the needs for the project very well, for example Personas [21]. Also, the project might have benefited from using a more strictly formal and scientific method for evaluating navigation methods.

Finally, the perhaps greatest challenge was the distribution of time into the separate iterations and prototypes. This is also one of the reasons later prototypes have relied more on evaluation than testing.

8 BIBLIOGRAPHY

1 Snyder, Carolyn. *Paper prototyping*. 2001.

2 Wharton, Cathleen, Rieman, John, Lewis, Clayton, and Polson, Peter. The cognitive walkthrough method: a practitioners guide. In Nielsen, Jakob and Mack, Robert L, eds., *Usability Inspection Methods*. John Wiley & Sons, New York, 1994.

3 Nielsen, Jakob. How to Conduct a Heuristic Evaluation. In Nielsen, Jakob and Mack, Robert L, eds., *Usability Inspection Methods*. John Wiley & Sons, New York, 1994.

4 Ware, C and Osborne, S. Exploration and virtual camera control in virtual three dimensional environmnets. (New York 1990), ACM.

5 Tan, Desney S, Robertson, George G, and Czerwinski, Mary. *Exploring 3D Navigation: Combining Speed-coupled Flying with Orbiting*. Microsoft Research, Redmond, 2001.

6 Darken, R. P and Sibert, J. L. A toolset for navigation in virtual environments. (New York 1993), ACM.

7 Burtnyk, N, Khan, A, Fitzmaurice, G, Balakrishnan, R, and Kurtenbach, G. Stylecam: interactive stylized 3d navigation using integrated spacial & temporal controls. (New York 2002), ACM.

8 Hanson, A. J and Wernert, E. A. Constrained 3d navigation with 2d controllers. (Los Alamitos 1997), IEEE Computer Society Press.

9 Hanson, A. J, Wernert, E. A, and Hughes, S. B. *Constrained navigation environments*. IEEE Computer Society, Los Alamitos, 1997.

10 Haik, E, Barker, T, Sapsford, J, and Trainis, S. Investigation into effective navigation in desktop virtual interfaces. (New York 2002), ACM.

11 Salomon, B, Garber, M, Lin, M. C, and Manocha, D. Interactive navigation in complex environments using path planning. (New York 2003), ACM.

12 dos Santos, C. R, Gros, P, Abel, P, Loisel, D, Trichaud, N, and Paris, J. P. Metaphor-aware 3d navigation. (Washington 2000), IEEE Computer Society.

13 Galyean, T. A. Guided navigation of virtual environments. (New York 1995), ACM.

14 Elmqvist, N, Tudoreanu, M. E, and Tsigas, P. Evaluating motion constraints for 3d wayfinding in immersive and desktop virtual environments. (New York 2008), ACM.

15 Abásolo, Maria J and Della, José Mariano. *Magallanes: 3D Navigation for Everybody*. University of the Balearic Islands.

16 Sundin, Martin and Fjeld, Morten. Softly Elastic 6 DOF Input. *International Journal of Human-Computer Interaction*, 25, 7 (September 2009), 647-691.

17 Kim, W. S, Tendick, F, Ellis, S. R, and Stark, L. W. A comparison of position and rate control for telemanipulations with consideration of manipulator system dynamics. *IEEE Journal of Robotics and Automation*, 3, 5 (1987).

18 Wickens, C. D. *Engineering psychology and human performance*. Harper Collins, New York, 1991.

19 McCrae, James, Mordatch, Igor, Glueck, Michael, and Khan, Azam. *Multiscale 3D Navigation*. Autodesk Research, Toronto.

20 Chittaro, Luca and Burigat, Stefano. 3D location-pointing as a navigation aid in Virtual Environments. (New York 2004), ACM.

21 Long, Frank. Real or Imaginary: The effectiveness of using personas in product design. (Dublin 2009).

APPENDIX A: USER TESTS

The tests have been performed following this script though small deviations have occurred depending on the application being tested. The tests have been performed following a task oriented talk-aloud pattern with supervised Heuristic Evaluation in the end.

1) The observer explains how the test will work; that the user will answer a few questions and then be given a set of tasks to complete and should speak aloud about what he/she is doing, the problems that arise and if he/she is uncertain or unable to proceed.

2) The user is asked the following questions:

Question 1:	Are computers a part of your professional working environment and, if so, in what form?
Question 2:	What programs do you mostly use?
Question 3:	What sort of computer do you mostly use; laptop or desktop?
Question 4:	Do you play games on the computer and, if so, which/what kind of games?
Question 5:	Do you play or have you played 3D games on the computer.
Question 6:	Do you have previous experience with Virtual Reality (except for games)?
Question 7:	Have you used Dynagraph's earlier applications?

3) If the original application is being tested and the user does not have previous experience with it, a quick overview of the controls and interface is given. If a prototype is tested the user may be told how to navigate, but only very briefly.

4) The user is given a set of tasks. If the observer wants to look at a special task more closely he can ask the user to repeat variations of that task. The navigation controls are mostly tested in the first task, so the user may be given several target locations to navigate to in order to test them well.

Task 1:	Navigate to position designated by observer (the observer points at a place in the scene or describes it to the user so the user understands where to go).
Task 2:	Rotate the camera around 360 degrees to get an overview of the area.
Task 3:	Place a viewpoint on the current position and rotation.
Task 4:	Navigate to ground level and follow a path/road designated by the observer.
Task 5:	Return to the position you had before the last task.
Task 6:	Navigate to the designated position and then jump to the viewpoint created earlier to get back (Only used if the user did not use the viewpoint to return in the previous task).
Task 7:	Rename the Viewpoint you just jumped to.
Task 8:	Jump to another viewpoint.
Task 9:	Delete the viewpoint you just jumped to.
Task 10:	Play any tour available.
Task 11:	Repeat the tour and look around while the tour is playing.
Task 12:	Pause the tour and navigate to designated position.
Task 13:	Resume the tour.
Task 14:	Start the tour from the beginning again.
Task 15:	Navigate to designated position.
Task 16:	Answer the question: At what side of designated building/object is the sun during a summer morning?
Task 17:	Increase the shadow opacity.
Task 18:	Alternative task 16B: Hide the toolbar to maximize the screen size (if using the original application).

Task 19: Measure the distance between two designated points.

Task 20: Answer the following questions: (If testing the original appl. or prototype 1)

i. How far is it between the measured points?
ii. How high is "Point A" from "Point B"?

Task 21: Jump to any viewpoint and save the view seen from there as an image.

Task 22: Change the appearance of a model.

5) The user is asked to comment on any part of the interface or controls and ask questions if there are any.

6) An informal interview is held with user discussing any interesting issues that may have arisen during the test or something the user commented on afterwards.

The results of the testing are presented in Appendix B.

APPENDIX B: TEST RESULTS

For result evaluation the test users have been placed into three different categories; novice, intermediate and expert. These categories do not describe the users experience or skill with the prototype or original application, but rather his/her skill and experience with computer software in general and 3D games in particular. The skills needed to be categorized as intermediate and expert are also consistently lower than in many definitions of the words, since the goal of the project is to design for beginner users. In other words, users labeled experts in this project would quite probably be labeled intermediate in other similar tests.

Novice: The user has very little experience with computer software in general and no experience in 3D games.

Intermediate: The user uses different forms of computer software daily and has limited experience with games, sometimes even 3D games.

Expert: The user has played several 3D games or is experienced in Dynagraph's old application.

NAVIGATION

The testing of the navigation techniques was mainly done through observing users navigate with different techniques and see which technique presented the least problems for the different users groups. Users were also asked which techniques they preferred if they were presented with several.

ORIGINAL APPLICATION NAVIGATION

This technique was tested with 9 test users (5 novice, 2 intermediate, 2 expert) and all but one (expert) exhibited difficulty changing between the different modes used in navigation. The same 8 users also had problems understanding that forward and backward movement only happened in the horizontal plane. The one person who exhibited no problem with this technique had previous experience with the original application.

MULTISCALE NAVIGATION

This technique was tested with 3 test users (2 novice, 1 expert) and all users found the technique stressful to use. The two novice users remarked that they felt they had little control over the navigation. Both novice users released the mouse in an attempt to stop the movement. All three users preferred the original application navigation technique over the Multiscale technique.

WASD CONTROLS WITH AUTOMATIC SPEED

This technique was tested with 10 test users (4 novice, 4 intermediate, 2 expert) and all novice users as well as one intermediate user exhibited difficulty using both hands for navigation. One intermediate user misunderstood the sideways movement action in the S and D keys and expected the camera to turn instead of strafe. Three of the remaining four users (1 intermediate, 2 expert) had experience with 3D games, one (expert) playing several hours a week.

CHASE CAMERA WITH AUTOMATIC SPEED

This technique was tested with 10 test users (4 novice, 4 intermediate, 2 expert) and all users found the controls easy to understand. Three novice users had problems with the movement speed when in dense areas, sometimes overshooting. One expert user called the technique “fun” to use. Two users (1 intermediate, 1 expert) preferred the WASD controls to the Chase Camera. Out of the 10 users 7 (3 novice, 2 intermediate, 2 expert) had tested the original application controls and all 6 users preferred the Chase Camera to the original application navigation technique.

HOVER CAMERA

Two versions of the hover camera were tested in two different navigation techniques. The hover camera tested with the Multiscale navigation scheme had a *position control* transfer function and followed the *scene-in-hand* principle. The hover camera tested with the WASD controls and the Chase Camera had a rate control transfer function and did not use the scene-in-hand principle.

The first version was tested on three users (2 novice, 1 intermediate). The novice users found the camera hard to control and understand, having to move the mouse too much. The intermediate user liked using it but also exhibited problems in moving the camera as desired.

The second version was tested with four novice users and they all appeared to be able to handle it without complication. One user remarked that it felt “nice” using the camera. One user used the hover camera to elevate the camera instead of traveling upwards or panning.

GRAPHICAL USER INTERFACE

The results from the graphical user interface testing are presented for each task the users were asked to complete. For each task a table is presented which gives details about how the different users performed the task. There are three “results” for each task labeled:

- **The user completed the task to satisfaction**: This means that the user completed the task within “reasonable time”. The user should not have had to spend too much time with trial-and-error actions. For this to apply the user must also understand the outcome of the task, i.e. be sure of his/her success.
- **The user completed the task with difficulty**: This means that the user completed the task, but took a long time doing so or succeeded more thanks to trial-and-error behavior than an understanding of the interface. As with the first result type, this also requires the user to understand the outcome of the action.
- **The user completed the task but did not understand the outcome**: This means that the user succeeded in the task but was not sure he/she had succeeded for lack of understanding of the outcome.
- **The user failed the task**: This means that the user needed instructions in order to complete the task.

Since the number of test subjects differed between the different tests the table figures are stated as ratios. Each table is also followed by comments about interesting user behavior.

The versions listed in the table below are the versions of the corresponding tools, so for the task “Playing a tour” prototype 2 refers to the second prototype of the Tour controls. In the viewpoint tasks, prototype 2 refers to the “tourist version”.

Task: Create a viewpoint									
Task Result	**Original Application**			**Prototype 1**			**Prototype 2**		
	Novice	Inter.	Expert	Novice	Inter.	Expert	Novice	Inter.	Expert
User completed the task to satisfaction	1/5	1/2	1/2	1/5	0/2	0/1	4/4	4/4	2/2
User completed the task with difficulty	3/5	0/2	1/2	2/5	2/2	1/1	0/4	0/4	0/2
User completed but did not understand outcome	1/5	1/2	0/2	0/5	0/2	0/1	0/4	0/4	0/2
User failed the task	0/5	0/2	0/2	2/5	0/2	0/1	0/4	0/4	0/2

Comments Original Application: Two users did not understand they had succeeded since they received no feedback. They understood when looking in the list.

Comments Prototype 1: Seven users did not find the tool menu. All of these users looked in the tool menu. One expert user saw the side menu button but did not connect it to the viewpoints.

Comments Prototype 2: One intermediate user wanted the window to be more transparent to be able to see the background.

Task: Jump to viewpoint									
Task Result	**Original Application**			**Prototype 1**			**Prototype 2**		
	Novice	Inter.	Expert	Novice	Inter.	Expert	Novice	Inter.	Expert
User completed the task to satisfaction	5/5	1/2	2/2	5/5	2/2	1/1	4/4	4/4	2/2
User completed the task with difficulty	0/5	1/2	0/2	0/5	0/2	0/1	0/4	0/4	0/2
User completed but did not understand outcome	0/5	0/2	0/2	0/5	0/2	0/1	0/4	0/4	0/2
User failed the task	0/5	0/2	0/2	0/5	0/2	0/1	0/4	0/4	0/2

Comments Prototype 1: Having found the side menu the act of jumping to a viewpoint was performed easily by all test users.

Task: Delete viewpoint									
Task Result	**Original Application**			**Prototype 1**			**Prototype 2**		
	Novice	Inter.	Expert	Novice	Inter.	Expert	Novice	Inter.	Expert
User completed the task to satisfaction	2/5	0/2	0/2	1/5	1/2	1/1	4/4	4/4	2/2
User completed the task with difficulty	0/5	0/2	0/2	1/5	0/2	0/1	0/4	0/4	0/2
User completed but did not understand outcome	2/5	1/2	2/2	0/5	0/2	0/1	0/4	0/4	0/2
User failed the task	1/5	1/2	0/2	3/5	1/2	0/1	0/4	0/4	0/2

Comments Original Application: Several users did not understand that they had successfully deleted the viewpoint since the name was still shown in the text field. One user wanted to choose the viewpoint after clicking the delete button and therefore removed the wrong viewpoint.

Comments Prototype 1: One user tried right clicking the viewpoint in order to remove it. Four users did not understand the delete button icon. One of these thought it meant “Close window”.

Comments Prototype 2: All testers but one intermediate user had participated in previous tests and therefore may have recognized the “Delete” method.

Task: Start tour									
Task Result	**Original Application**			**Prototype 1**			**Prototype 2**		
	Novice	Inter.	Expert	Novice	Inter.	Expert	Novice	Inter.	Expert
User completed the task to satisfaction	3/5	2/2	1/2	4/5	1/2	1/1	4/4	4/4	2/2
User completed the task with difficulty	1/5	0/2	1/2	1/5	1/2	0/1	0/4	0/4	0/2
User completed but did not understand outcome	1/5	0/2	0/2	0/5	0/2	0/1	0/4	0/4	0/2
User failed the task	0/5	0/2	0/2	0/5	0/2	0/1	0/4	0/4	0/2

Comments Original Application: Two novice users did not see the play button or did not understand that it was a button.

Comments Prototype 1: Two users found the play controls hard to see.

Task: Pause/Stop/Resume tour									
Task Result	**Original Application**			**Prototype 1**			**Prototype 2**		
	Novice	Inter.	Expert	Novice	Inter.	Expert	Novice	Inter.	Expert
User completed the task to satisfaction	5/5	2/2	2/2	3/5	1/2	1/1	4/4	4/4	2/2
User completed the task with difficulty	0/5	0/2	0/2	2/5	1/2	0/1	0/4	0/4	0/2
User completed but did not understand outcome	0/5	0/2	0/2	0/5	0/2	0/1	0/4	0/4	0/2
User failed the task	0/5	0/2	0/2	0/5	0/2	0/1	0/4	0/4	0/2

These tasks were evaluated together since users exhibited the same behavior for all of them. If a user found and understood the play control, these controls were simple to use in all versions.

Comments Original Application: all users completed these tasks easily, since they had found the controls when starting the tour.

Comments Prototype 1: Two novice users did not see the controls, even though they knew about the play control. One intermediate user was confused about the difference between pause and stop.

Task: Save a screenshot									
Task Result	**Original Application**			**Prototype 1**			**Prototype 2**		
	Novice	Inter.	Expert	Novice	Inter.	Expert	Novice	Inter.	Expert
User completed the task to satisfaction	3/5	1/2	2/2	1/5	0/2	0/1	4/4	4/4	2/2
User completed the task with difficulty	0/5	1/2	0/2	1/5	0/2	1/1	0/4	0/4	0/2
User completed but did not understand outcome	1/5	0/2	0/2	0/5	1/2	0/1	0/4	0/4	0/2
User failed the task	1/5	0/2	0/2	3/5	1/2	0/1	0/4	0/4	0/2

Comments Original Application: One novice user wanted to see the file size, since that was the user's way of determining quality. Two users did not see the settings for setting format and size. One user was unsure what the formats meant.

Comments Prototype 1: Three novice users did not understand the format and size options. One intermediate user understood the formats but was unsure what the sizes meant. Another intermediate user misunderstood the task and created a viewpoint instead.

Task: Change a model alternative setting						
Task Result	**Original Application**			**Prototype 1**		
	Novice	Inter.	Expert	Novice	Inter.	Expert
User completed the task to satisfaction	0/5	0/2	0/2	0/4	1/4	2/2
User completed the task with difficulty	1/5	1/2	1/2	0/4	0/4	0/2
User completed but did not understand outcome	0/5	1/2	1/2	4/4	3/4	0/2
User failed the task	4/5	0/2	0/2	0/4	0/4	0/2

Comments Original Application: Most users had severe problems finding the tool. Two of the four that did succeed did not know if anything had changed.

Comments Prototype 1: Six users were confused by the fact that they could not see the changes. Two users were confused by the label and button texts.

Task: Change the time for the sun light						
Task Result	**Original Application**			**Prototype**		
	Novice	Inter.	Expert	Novice	Inter.	Expert
User completed the task to satisfaction	2/5	1/2	0/2	4/4	4/4	2/2
User completed the task with difficulty	1/5	1/2	1/2	0/4	0/4	0/2
User completed but did not understand outcome	0/5	0/2	0/2	0/4	0/4	0/2
User failed the task	2/5	0/2	1/2	0/4	0/4	0/2

Comments Original Application: Six users had problems finding the update button or realizing it was a button. Three of those needed help to see it.

Task: Measure a distance									
Task Result	**Original Application**			**Prototype 1**			**Prototype 2**		
	Novice	Inter.	Expert	Novice	Inter.	Expert	Novice	Inter.	Expert
User completed the task to satisfaction	2/5	1/2	0/2	1/6	2/3	1/2	0/1	1/1	0/0
User completed the task with difficulty	1/5	1/2	2/2	2/6	1/3	0/2	1/1	0/1	0/0
User completed but did not understand outcome	2/5	0/2	0/2	3/6	0/3	1/2	0/1	0/1	0/0
User failed the task	0/5	0/2	0/2	0/6	0/3	0/2	0/1	0/1	0/0

Comments Original Application: One novice user wanted to be able to measure volumes, areas etc. Two of the novice users had severe problems understanding the presented data.

Comments Prototype 1: Five users did not understand the lines used for presenting height and horizontal length.

Comments Prototype 2: One novice user missed the target surface when trying to end the measurement. The user did not understand the problem and tried again.

APPENDIX C: COGNITIVE WALKTHROUGH REPORTS

During the cognitive walkthrough only potential usability issues were noted, thus if parts of the graphical user interface was not deemed to have a potential usability issue it is not listed here.

Main interface - prototype 1

The maximize button can prove a potential problem. Its position at the lower-right can cause the button not to be seen by the user. Also its icon should be understandable by most people who have used any type of media player on the internet such as YouTube since the icon is almost standard for maximizing the player. However if the user have not used such, he/she might have a problem understanding it. The gravity button is placed in the menu which is not natural to the user. The user will not know he is to look in the tool menu in order to find the button, or he will look for a settings window.

Viewpoints - prototype 1

The open-button might prove to be too small which would result in the user not finding it or the user taking too long to find it.

Tours - prototype 1

The open-button might prove to be too small which would result in the user not finding it or the user taking too long to find it. Play, pause and stop in the side menu suffer from bad affordance which might result in the user not understanding that the buttons are clickable. Also the buttons are too transparent and hard to see.

Save image - prototype 1

The choice of image size; small, medium and large might prove a problem to the user. He or she might not understand the implications of these alternatives. Using buttons as toggles instead of more recognized forms may result in the user not understanding their function.

Model alternatives - prototype 1

The users might not notice if the model alternative is being active or not, the only confirmation is the switch of the active/inactive label. If the user is stationed at the wrong place in the VR world he/she will not notice the difference in the environment. Another potential problem is the status label and the activate/deactivate button, since the label shows the current status of the alternative and the button shows the action needed to change e.g. the model is inactive then the label says inactive and the button shows "Activate". This might confuse the user as whether or not the alternative actually is active.

Model alternative - prototype 2

It is not apparent to the user that he/she can click the image in order to jump to the model alternative, thus the image suffers from bad affordance. The active/inactive button can present a potential problem as the button's text shows the current status of the alternative rather then what the button actually does. The user might not understand that the button is used as a control to toggle the alternative on and off.

Settings window

Users might be confused since changes in the settings only take effect after the user clicks OK. The user is not notified of this when making a change in the settings.