CHALMERS

3D Window Manager Prototype

Master of Science Thesis in the programme Interaction Design

Andreas Jonsson
Marcus Järbratt

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden June 2009

3D Window Manager Prototype
Andreas Jonsson
Marcus Järbratt

Examiner: Olof Torgersson

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

Cover:
A screenshot of the resulting window manager prototype.
See Section 6.1.5 and Figure 17(a).

Department of Computer Science and Engineering
Göteborg, Sweden June 2009

Abstract

The standard graphical user interfaces on leading operating systems today all use a two dimensional approach for interaction and visualization. Handling several applications using this approach often leads to a cluttered work space which is hard to manage efficiently. Research within the 3D window management area has been performed and many products exist today which utilize 3D in one way or another, but none of these have had any impact on how windows are managed in leading operating systems.

The main goal with this thesis was to implement a prototype which utilized 3D in order to manage open windows in a structured way. The result is a system where the user can interact with windows in a 3D world. These windows can be placed and grouped on certain designated areas, which the user is able to navigate between quickly along predefined paths. In order to work with windows they are brought up from the 3D environment to a conventional two dimensional work space.

Different users that tried out the prototype encountered no big issues with how the prototype solved navigation and interaction with objects in a 3D environment. While some of them appreciated the product as a whole, some could not see how this would benefit them during their daily computer usage, although they saw advantages with some of the functionality.

Sammanfattning

De grafiska användargränssnitt som används i de ledande operativsystemen idag använder ett två-dimensionellt tillvägagångssätt för interaktion och visualisering. Att hantera flera applikationer på detta sätt leder ofta till en rörig arbetsyta som är svår att arbeta med effektivt. Forskning inom fönsterhantering i 3D har utförts och många produkter finns i dagsläget som utnyttjar 3D på ett eller annat sätt, men ingen av dessa har haft något genomslag för hur fönster hanteras i de ledande operativsystemen.

Huvudmålet med denna rapport var att implementera en prototyp som utnyttjade 3D för att hantera öppna fönster på ett strukturerat sätt. Resultatet är ett system där användaren kan interagera med fönster i en 3D-värld. Dessa fönster kan placeras och grupperas på särskilda ytor, som användaren snabbt kan navigera mellan längs förbestämda banor. För att arbeta med fönster tas de upp från 3D-miljön till en konventionell två-dimensionell arbetsyta.

En grupp användare som prövade prototypen stötte inte på några större hinder med hur prototypen löste navigering och interaktion med objekt i en 3D-miljö. Medan några av dem uppskattade produkten i sin helhet, hade en del svårt att se hur detta skulle gynna dem i sin dagliga datoranvändning, även om de såg fördelar med delar av funktionaliteten.

Preface

This is done as a master thesis on the master program Interaction Design of Chalmers University of Technology. We want to thank our supervisor, Olof Torgersson, for giving good feedback and approaching the prototype with a fresh perspective. We would also like to send our regards to those trying our prototype offering their time to give us valuable feedback.

Contents

1 Introduction

Most of the interaction with computers today is made using a graphical user interface (GUI) in a two dimensional (2D) manner. One of few exceptions is computer games where three dimensional (3D) worlds have been successfully used for a long time.

The operating systems (OS) and the way they present the user with the applications works basically the same as they did a decade back. The windows are placed on top of each other on the screen, and when the user exceeds a few open programs, they are cluttering up the view, and it is hard to separate which window you want, especially if you have a lot of running instances of the same application.

UNIX, among other systems, has had support for different desktops for a long time, which enables you to divide the windows and elements on different views. One of the first window managers (WMs) introducing this concept to the public was the Solbourne Window Manager [15, chapter 6.1]. This is one enhancement where you can divide up different tasks on different desktops, but working with one task can still mean having a lot of different applications running to achieve this.

Some improvements to use multiple applications in todays operative systems has been made, e.g. by giving an opportunity to quickly switch between open applications with keyboard shortcuts. These are often tedious to use though when the number of open windows increases, requiring the user to scroll through a long list or large amount of thumbnails to find that window he was searching for. A process which still creates unnecessary pauses in the work flow.

Within the research area of interaction there has been some tests using 3D, but since the mouse only provides input from two dimensions, the interaction can easily confuse the unaccustomed users who are not using 3D interaction software on a regular basis. Still, the third dimension may open up for new functionality and may very well contribute for a more effective computer use.

1.1 Aim

The goal with this thesis is to implement a prototype of a 3D interface system one can use to simulate common OS functionality. This prototype does not target any specific platform or OS, and the intention is to make a prototype from scratch designing and implementing all parts needed for 3D graphics and 3D interaction. Using this prototype it is intended to test whether people are able to perform interaction which feels as intuitive and

straight forward to work with as the 2D interfaces standard on computers today. The main focus will be on managing windows in a 3D space, while keeping the individual windows' 2D representation, and addressing common window and desktop management issues.

Previous research within relevant areas and already existing systems will be used for inspiration. Inspiration for new interaction is hard to find in the systems today, since they are all influenced by the regular expected 2D functionality. By looking at different other software areas, like computer games, there is hope to find exciting and useful new ways of interaction.

Finally, the primary aim with the prototype is not be a replacement for average computer users. The functionality is foremost intended to ease the work flow for users who multitask using several programs simultaneously. It focuses on increasing productivity while minimizing time wasted on tedious tasks such as finding the right window or navigating interfaces. Still, the intended target group is quite narrow so it will be interesting to see how people with different computer usage and experience could benefit from the ideas as well.

1.2 Delimitations

The amount of work this thesis corresponds to makes it hard to implement and finalize a fully functional product. Instead it was decided to make a prototype which has the needed key functionality to try the concepts of 3D interaction and window management.

One part which was decided to skip, at least initially in the development process, was to connect the system to an actual operating system. Different operating systems would in that case need to be studied in order to be able to do this, leaving less time to spend on other key functionality such as 3D rendering and interaction.

A 3D rendering engine can be very complex and is not something that is easily implemented. The intention is not to make a 3D rendering engine equipped with a lot of state of the art techniques that are not needed for the purpose of this project anyway. Instead a basic rendering engine with functionality sufficient to fulfill the needs and requirements of visualizing the functionality in the prototype will do.

2 Related work

There are many different aspects of this thesis regarding interaction, 3D graphics, and window management. A lot of interesting research has been done in these areas, and many commercial products as well as various freeware are of interest and a source of inspiration.

2.1 Products & Prototypes

Today's leading operating systems using a GUI for interaction all share the same main concepts. 2D is used almost exclusively and a lot of functionality has been used for many years and has not evolved to any appreciable extent.

There is often a desktop behind everything else on the screen. On this desktop one can place icons for shortcuts, files etc. When a window is opened it gets in the way of the desktop and in order to get access to the desktop again you either have to move around any open window that is in the way or minimize them.

Open windows are often placed in a taskbar where you can click with the mouse in order to switch among them. This tends to work well as long as there are not too many open programs. When reaching above a certain number, the titles of the open programs are no longer visible and the only way to distinguish among the open windows is by looking at which icon the program has. This leads to problems when having several windows of the same application running.

In Microsoft (MS) Windows, among other systems, *Alt+Tab* is a very useful keyboard command combination to quickly switch among open programs for a keyboard driven user. By using this key combination an overview of the open windows appear on top of the screen where the user can choose which program to switch to. In MS Windows XP this has very limited functionality. For each open window it only displays window title and the application icon giving the user a poor overview of the open windows. Another drawback is limitations in how you select a window since the user is only allowed to go forward/backward one step at a time by pressing the *Tab* key multiple times while still holding down the *Alt* key. In MS Windows Vista this functionality is slightly improved by showing a preview of each window, making it possible to select a window by clicking its preview with the mouse, as well as selecting window with arrow keys. The user can also by pressing the keyboard combination *Win+Tab* get an overview of all open windows in 3D instead.

The Mac OS X also has an interesting feature called Exposé [17]. By pressing *F9* the user can instantly get all the open and unhidden applications side-by-side. By clicking on any item this is brought to the front, or

Figure 1: Screenshot from Mac OS X getting an overview of the running applications. [33]

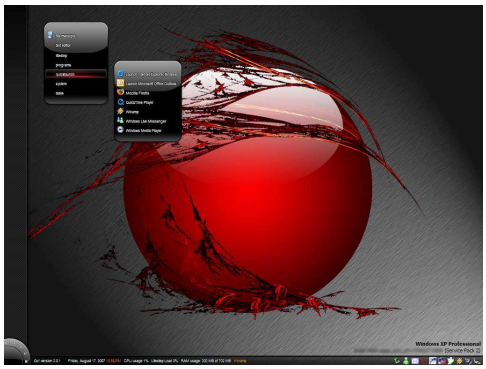


Figure 2: Screenshot from a customized desktop with the Litestep window manager. [34]

Figure 3: Screenshot from a desktop with the BumpTop extension. [35]

Figure 4: Concept image of a desktop with the 360Desktop extension. [36]

by pressing F9 again, the windows are returned to their original scale and position. See Figure 1 for reference.

There are a number of different window manager replacements for MS Windows, some of them are just to change the appearance of different elements of the GUI, like changing the images used on buttons and menus for instance. There are also some programs that can be used to replace specific elements. ObjectDock [22] for instance, can be used to replace the start menu, taskbar and system tray. Other existing solutions change how you interact with the OS on a more extensive way.

LiteStep [16] is a window manager for Microsoft Windows. It centers on the idea that the users can script any interface they want. By a wide variety of modules, one can by Lua scripting or by creating C++ modules

get any interface wanted. Some elementary GUI elements are forced, such as the action buttons for the windows, but one can add desktops, different controller interfaces for running applications into the GUI etc. Since the themes are script-based, there is no enforced compilation for the packages to run on the LiteStep core, which means that unwanted or badly performed functions in a theme can be altered to better suit the user requests. See Figure 2 for reference.

BumpTop [5] is a physics-based 3D desktop for MS Windows. It is not really a window manager, but it has a lot of new ideas that still makes it interesting as an inspiration for this work. It has focused a lot on interaction of icons and files on the desktop. Files can be grouped together, piled or ordered after customized routes over the desktop. There is also an ability to attach different gadgets to the desktop walls or floor. See Figure 3 for reference.

360Desktop [1] has the same functionality as any ordinary window manager, with the only difference that instead of having different desktops that can be switched between, there is an extended desktop. The windows can be placed on any part of this surface, and the orientation between them is done by rotating the desktop space. The extension also allows for gadgets to be placed wherever they are wanted along the surface. See Figure 4 for reference.

There is also a wide range of applications which all have the same functionality, the classical cube. One of these applications is CubeDesktop Pro [8]. The idea is that each side of a cube represents an individual desktop. The only functionality the cube has is for the user to easier locate the different desktops without losing orientation. The cube could be replaced by any other geometrical shape, although the user orientation might get lost if the available surfaces grow to large. By having multiple desktops, one can easily move different application windows to different areas to decrease cluttered windows. See Figure 5 for reference.

Metisse [19] is a 3D based window manager for the X window system. It is, at least officially, only available for the Linux-based OS Mandriva. Even though the impression still is 2D, it supports a lot of easy-to-use functions where the user can fold away windows covering the screen, as well as orienting and rotating them in any possible way. A lot of the functionality seen in the demonstration videos from their website is of a great interest for the development of this system, amongst others their view with the desktops side-by-side. See Figure 6 for reference.

Figure 5: Screenshot of an operating system with the cube extension CubeDesktop Pro. [37]

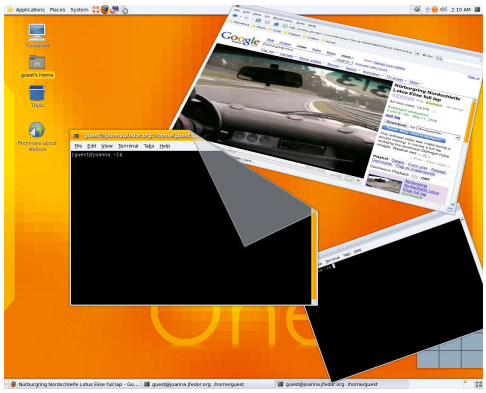


Figure 6: Screenshot of Mandriva One, running the Metisse window manager. [38]

2.2 Research

When implementing a 3D window manager prototype different areas of research are of interest. Three areas have been of greater interest than others: approaches to 3D window managing, different ways of interaction in 3D, as well as orientation and spatial cognition in a 3D environment.

2.2.1 Window Managing

There is not very much research to be found regarding window managers using 3D interfaces. One paper from Microsoft research department is frequently cited though. They have made some interesting research within the area and produced a 3D window manager they called *Task Gallery* [27]. In Task Gallery applications and functionality is placed within a 3D environment which the user is able to walk around in. The environment is designed to look like an indoor art gallery. In this gallery applications can be grouped together as tasks and be placed on walls, ceilings, floors and notice boards in the scene. See Figure 7 for an example. When using Task Gallery the primary use of the 3D environment is to manage tasks. When one wants to work with an application it still uses the 2D approach.

One point with Task Gallery was to study the effects of human spatial cognition and perception while utilizing the 3D hardware in computers. The primary purpose with this was to ease the process of task management and ability to compare multiple windows. The results of the user tests are very interesting showing that the test subjects did remember very well where in the 3D scene they had placed different tasks. It was also found that the test

Figure 7: Screenshot of Task Gallery in use. Active windows have been grouped as tasks and been put on areas in a 3D room. [39]

subjects did not have any problems adjusting to the approach with a 3D world concept.

3Dwm *(Three-Dimensional Workspace Manager)* [10] is another research project about extending the 2D user interface into 3D. The report covers a project of implementing a research and development platform for 3D user interfaces in applications. Compared to Task Gallery, which still uses 2D user interfaces for running applications, this project is more about replacing the 2D user interfaces completely with user interface components in 3D. To succeed with this some tools for creating custom 3D components are incorporated.

WindowScape [31] is a project that addresses some issues connected to task management by exploiting users' spatial and visual memories. One of its main features is to use miniature snapshots of windows to more easily find them among others. It also introduces an interesting approach to grouping programs as tasks using a history metaphor. Depending on which programs that are used at a certain time they can be grouped as a task and a snapshot of that workspace is stored in a history timeline. A task and its corresponding programs can then be resumed from the timeline.

Later versions of MS Windows among other operating systems, uses a taskbar to keep and give access to open windows (see Section 2.1). Microsoft published a paper where the limitations of the taskbar were addressed and the taskbar was evolved into the GroupBar [29]. The paper mentions the need for a new way of handling open applications, due to the difference in work structure that has evolved since the size of screens have grown, which has lead to people using a larger number of simultaneous applications multitasking.

2.2.2 Interaction With 3D Interfaces

Interaction in 3D can cause problems for the user and many interaction designers have made research in this area. Alan Cooper is well known in the area of interaction design. In his book *About Face - The Essentials of Interaction Design* [7] some issues are addressed and some guidelines are presented on how to design for better interaction in 3D. He states that a 3D space projected on a 2D screen introduces problems for the human to perceive the scene correctly. An issue that is brought up is movement. Unconstrained movement in 3D space is for instance something that humans are not used to and should be avoided.

Ben Schneiderman is another well-reputed man in the area of interaction between humans and computers. In the paper *Why Not Make Interfaces Better than 3D Reality?* he discusses benefits and drawbacks of using 3D instead of 2D in interfaces [28]. He states some good points, some more obvious than others, although not necessarily unique to 3D interaction. The advices are amongst others minimizing the number of steps for users to accomplish their tasks and keep the text readable by having size, contrast and tilt in mind, as well as trying to prevent the possibility of making errors by only making the user able to perform allowed actions and having access to undo/redo actions. He also wants to avoid unnecessary visual clutter (distractions, reflections, contrast-shifts etc) and simplify object movement by e.g. docking, predictable paths and limiting rotation. Some "enhanced features" as they are called in this paper are also mentioned, such as providing an overview of the scene and allowing teleportation in order to quickly and easily move around. Another advice is to give the possibility to see beyond or through objects in order to prevent occlusion of unneeded items and having the functionality of dynamic queries in order to filter out unneeded items rapidly.

One of the benefits of using 3D compared to 2D is that you can place objects more freely in order to save screen estate. When using 3D you have an opportunity to rotate objects such as windows, menus, and other GUI related visual parts in relation to the viewer in many different ways. An advantage of rotating an object away from the viewer is the fact that it will occupy less screen estate. The drawback on the other hand is that it will get distorted due to the 3D perspective of the scene. Some research in this area has been published that shows that text takes longer time to read when it is distorted in this manner [14]. Hence objects which display important information should be rotated carefully away from the viewer if rotated at all.

2.2.3 Spatial Cognition

In the systems today the benefits of human spatial cognition comes in use when people who use their desktop to keep shortcuts and files tend to group and place icons in a way that they know exactly where each icon is. The result is that they more easily and quicker can find an icon when they need it without any need to search through lots of icons.

In the paper *Data Mountain* from 1998 by Robertson et al. [26] they have made some tests on how spatial cognition can be utilized by introducing a different approach to storing and accessing information. The Data Mountain is a 3D document management system which in this paper was implemented as a prototype for managing web page bookmarks. The idea is that small thumbnails of web pages are placed on a tilted surface in a 3D scene. These thumbnails can be moved around freely on this surface. The thumbnails follows the tilt of the plane making them appear smaller the further back they are placed. This functionality gives an opportunity to group similar objects together or placing less important objects further away than the ones more frequently used.

During the user tests they tested how easy it was for the user to find a particular bookmark using Data Mountain compared to a traditional menu which lists all web pages top down. Based on this they summarized and compared how long the reaction time was to find bookmarks, how many incorrect retrievals that had to be done before finding the right one, and how many bookmarks that were not found at all within a two minute limit. The users testing the Data Mountain had better results on all of these points overall. They found them faster and some even remembered their location months after the test.

The research of Data Mountain is cited in many more recent publications. Andy Cockburn and Bruce McKenzie [6] took the tests further and tested three different approaches of data visualization called 2D, $2\frac{1}{2}$D, and 3D. In

(a) 2D (b) $2\frac{1}{2}$D (c) 3D

Figure 8: Three different approaches of visualizing stored data with spatial cognition in mind. [6]

all three of these approaches the bookmark image is faced towards the viewer without any rotation. The difference between them is how the bookmarks are placed and moved around. For 2D and $2\frac{1}{2}$D the bookmarks are placed on a plane. For 2D the plane is aligned along the screen and for $2\frac{1}{2}$D the plane is tilted away from the screen. For the 3D approach the bookmarks are placed freely in space. See Figure 8 for reference. In this paper they test response time for bookmark retrieval with different amount of bookmarks open. Three different levels were tested which they referred to as sparse, medium, and dense. Slightly different results could be found between the three approaches. Access time using the 2D approach increased very much for each step. For $2\frac{1}{2}$D the access time remained almost the same with only slight increase for each step. The approach with bookmarks placed free in 3D worked well for sparse but had a large increase for medium and dense.

2.3 3D Graphics - An Introduction

The process of generating a 2D image from a 3D scene is called rendering. The rendering process is performed by the rendering pipeline. The rendering pipeline can be conceptually divided into three parts: *application*, *geometry*, and *rasterizer*. While the application stage is always performed on the CPU the other two stages are preferably performed on the GPU.

The application stage is where the software runs. The software holds and controls all objects in the 3D scene in *world space* coordinates. All objects that are rendered consist of connected triangles. Each triangle consists of three positions in the 3D space called *vertices*. To determine from which position and what direction to render the scene a virtual camera is placed in the 3D world. The objects in the world that are rendered are the objects that are within the view frustum of the camera (see Figure 9). If the view frustum is seen as a cut-off pyramid the top of the frustum is called the near clipping plane and the bottom the far clipping plane. The camera's position is where the tip of the pyramid would have been if the pyramid would have been intact. The angle of the view frustum depends on the field of view setup for the camera.

When it is time to render the scene the application stage feeds the geometry stage with information about all objects in the scene and the location and direction of the camera. See Figure 9 for a visualization of the most important steps in the rendering process. All vertices in the scene are at this point transformed from global world space coordinates into *camera space* coordinates, which are their positions in the camera's local coordinate system. After some processing a *projection* is performed which transforms the view frustum in camera space into a unit cube, a space with the minimum and maximum points of -1 and 1 respectively. When this is done all primitives'

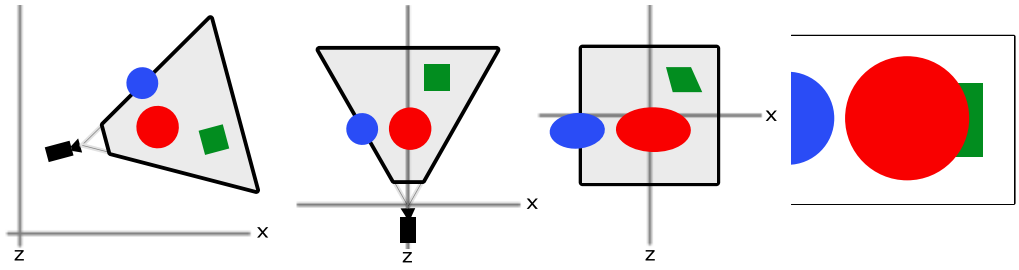


Figure 9: Visualization of four big steps in the rendering process seen from above. From left to right: camera with its view frustum and objects in world space, objects in camera space, objects in unit cube after projection, and the resulting image.

that are outside the unit cube are clipped and the remaining primitives x- and y-coordinates are mapped to match the position on screen. These primitives are sent to the rasterizer stage. At this stage each pixel in a color buffer is filled with the correct color, which depends on the position of the primitives from the previous stage. [2, chapter 2]

3 Task Analysis

The primary purpose with this system is to enable anyone using a computer to have a better structure and overview of running applications giving them easier and faster access when switching among them. The typical user who would benefit the most from this system is a person who needs to multitask by having several applications running at once, and perhaps even switches between different work tasks during the day.

The resulting product of this paper is not a fully working window manager connected to a running operating system. Instead it is a prototype written with the primary goal to test a new approach to window managing. The systems architecture is strongly inspired by rendering engines common in computer games today, which are capable of rendering 3D scenes to the computer screen using hardware accelerated graphics.

This prototype is based on a slightly different conceptual approach than the ones you find in other common operating system window managers. Even if the functionality is mainly the same as other window managers have, the way you interact with the computer is slightly different. The main issues which this thesis is trying work around have evolved from personal usage and experience of insufficient window managing possibilities in common operating systems. The different concepts and ideas to use in this prototype have evolved from inspiration received from a combination of successful parts of existing systems as well as various third-party solutions which also have some of these issues in mind.

The functionality of this prototype is not something which in the first place is intended for the average computer user. It is intended to make the work easier for those users who use several programs simultaneously and have a need to quickly being able to switch among different applications by not having to spend any more time than necessary searching for a particular application.

Even if the initially intended target group is rather limited it is still interesting to see how people from different user type groups will respond to this solution. With this in mind interviews with people with different computer backgrounds are planned, as well as some tests to see how they respond to these concepts and ideas.

The biggest difference with the ideas of window management in this prototype and in operating systems today is the use of 3D compared to 2D. By using 3D you get another dimension of screen estate to place valuable information in. On the other hand, when you add another dimension of complexity to the interaction, this can possibly make it harder for the user to feel comfortable when giving input. Especially since some interaction will

be made in 2D and some in 3D. It will also be interesting to see how different kinds of users will respond to this type of interaction.

The concept of having an area, such as the desktop, behind everything else that you work with leads to a few interaction related issues. One part of this prototype was to skip the traditional desktop metaphor and instead incorporate the benefits of the desktop into widgets placed on a widget area. This area is intended to be placed on top of all running applications when activated instead of lying behind at all times. Something interesting with this point is to find out how much people actually use their desktops and what functionality they make use of.

3.1 Implementation Requirements

C++ is the main programming language which is intended to be used in order to implement the prototype. It will be used for the program structure, the 3D graphics programming, as well as the interaction needed. The major benefits of C++ are its efficiency on various platforms with low-level capabilities as well as its extended object oriented capabilities.

The system will have different layers of functionality. On top there will be the interaction layer which the user comes in contact with when communicating with the prototype. This layer needs to get information from the operating system about the current programs and status. This information about the operating system and the location of different windows needs be sent to the graphics hardware in order to be rendered to the screen.

3.1.1 Interaction

This level of the system is the layer that the user sees and interacts with. This layer needs to be connected to an underlying layer that keeps track of all running applications. Given the time frame, the prototype will not be connected to the underlying OS, but will instead be fed with information from a dummy layer, a dummy OS, keeping record of fake applications.

As the prototype will not actually connect with the OS, the application must create a window and render a 3D scene in this window. Input from connected input devices must also be captured within this window.

As input into the system the traditional mouse and keyboard are seen as the primary devices, even if some other type of input devices might be interesting to try out. The reason for this is that this prototype is intended to be used with computers as they are used today and with the input devices people already are familiar with. To force people to learn using new tools in order to be able to handle a new system seems to be a too big step since

the primary objective with this system is to make the work flow with current work stations smoother and more efficient.

The mouse is intended to be used in order to handle windows in different ways both in 2D and 3D. Functionality which is seen as interesting to implement is to move, rotate and scale elements such as windows in both 2D and 3D. In 2D this is fairly straight forward but in 3D it can be done in different ways, either completely free or with some type of constraint. In order to find which approach to use different constraints need to be implemented in order to try what feels intuitive and is desired.

The ideas of the presence of windows that are placed in the 3D space is to either place them completely free or to attach them to a deskspace. A deskspace is a concept in this prototype which is an area that is placed in a 3D world containing visual representation of running applications. In order to find a certain application among the ones that are open and running an idea is to implement a search function that is easily accessible and gives a good overview of matching applications by fading out the applications that do not match what the user types.

An important feature for visual feedback in any GUI is the mouse tooltip. In order to have greater control over the prototype usability, the tooltip functionality will be needed to get implemented in the prototype.

In order to manage running applications in a 3D world some kind of framework needs to be created. This framework needs to handle window positioning and movement for instance. How this is realized depends on other design decisions.

The different elements that one can interact with in the 3D environment, such as the windows, need some sort of interface that lets the user know what possible actions there are. To respond correctly input must also be interpreted and forwarded to the window and its different interface parts when necessary.

3.1.2 Rendering engine

The rendering engine of the system must be able to keep the current state of all visible objects as well as being able to visualize them on the computer screen. Internally this means that the position and orientation of all objects needs to be stored. These properties need to be stored in a way that they can be easily modified when the user manipulates the state of any object in the 3D world.

To be able to render any object to the screen the system must hold a space representation, including interpreting position and orientation, which is compatible with the one used by any chosen rendering system.

In order get the different objects into the system easily, the system needs to be able to load and read different kinds of files. Examples of these would be image files for textures and text files with information about objects' representation primitives.

Similar to many computer games, the intended design of the prototype is to have objects in a 3D world as well as objects in 2D "in front of" the 3D objects. To be able to display information to the user, a vital functionality is to be able to render text on the screen.

In order to interpret the mouse actions correctly it must be possible to determine over which object the user is currently holding the mouse. For the objects in 2D this is a trivial task comparing coordinates, but for the 3D space some more advanced algorithms needs to be implemented.

Visual effects used correctly can be powerful and important to visualize different states of an interface. If and how effects will be used in the prototype is still unclear, but adding support for it is considered necessary.

4 Method

The implementation process was decided to be carried out in an iterative manner. The reason behind this decision was that it was seen as a good approach to have a working version of the prototype at all times, even if it was a prototype with limited functionality. With this approach it was easy to plan and implement new additions to the system step by step.

4.1 Project plan

All the details about the project are hard to know at an early planning stage, but the different deliverables can be roughly estimated and planned according to an approximated time plan.

In order to make the different parts of the tasks easier to grasp and to get a good structure of the work needed to be done, it was decided to break down the task into milestones. A milestone is the end of a developing stage where a significant deliverable is delivered [30, chapter 5.2.2]. In this case three big steps in the development cycle of the prototype were selected as milestones. These were: basic rendering and user input, some basic testable functionality and trying these on a test group, and finally adapting the system according to the feedback and polish the functionality.

To complete the first milestone and achieve basic rendering and user input, some more detailed subgoals had to be fulfilled. There has to be some initial research of the systems today, both default and customized products for different operating systems. There is also a need of creating a basic underlying structure of the prototype. A window should render a 3D scene and take input from both mouse and keyboard, both navigation and the possibility to pick objects in space using the mouse. Finally, the 3D scene needs to be fed with application information from a dummy OS as well.

The second milestone was decided to be a program with enough functionality to be able to perform some tests on a test group. The result from the first milestone will probably need some correction and fixes, so that most of the bad design ideas can be caught early and not cause problems later on in the development process. The next step will be to decide what kind of functionality to implement into the prototype, i.e. which functionality that feels relevant enough to spend time on investigating further. This functionality then needs to be implemented and a suitable test group needs to be found before the next milestone.

The third and final milestone was decided to be a working version of the prototype where the feedback from the user tests was taken into consideration. The feedback from the tests needs to be weighted to decide what is

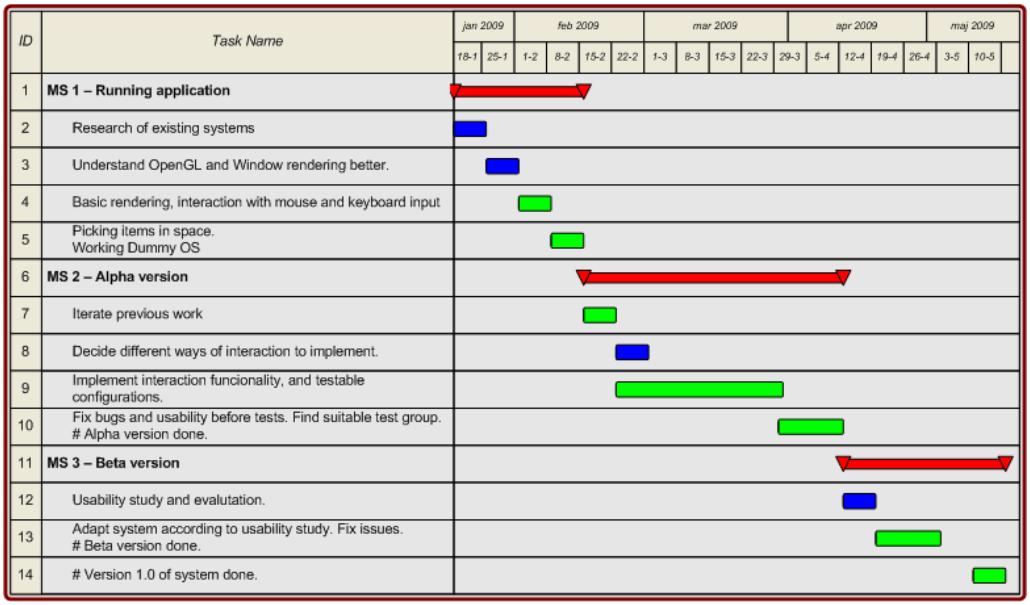


Figure 10: Rough planning of the project's different implementation phases with milestones and subtasks.

possible to implement given the remaining time, and then adjust the system according to these ideas, as well as correct any other functionality not working properly. Each of these milestones and their respective subtasks can be seen in relation to each other in the Gantt chart in Figure 10.

5 Realization

This chapter describes the process of implementing the prototype in chronological order. Motivating important design decisions and describing different problems along the way.

5.1 Milestone 1

The deliverable for this milestone was a running application with the most fundamental functionality implemented. The goal with this milestone was to look at relevant previous research and existing products, as well as connecting input devices to a running window in a MS Windows environment, and in this window render the content of a simple 3D scene using the 3D graphics hardware.

5.1.1 Research on Existing Systems

As a starting point, it was decided to examine what existing functionality in today's operating system that could be reused. It would both fill the functionality as well as ease the transition from the normal window management. The application window actions (minimize, restore/maximize and close) is something that has always been working very well.

Both the start menu and the system tray are areas that are not that frequently visited when the desired applications have been started. The screen space they require is small, but by being able to hide them when they are not used valuable screen space is made available. This type of functionality in today's OSs can be found in MS Windows where these areas can be hidden until the mouse cursor is hovered upon the lower edge of the screen.

The multi desktop support was seen as a requirement for this prototype. It is one way of dividing different task-dedicated applications and is decreasing the windows cluttering the screen. This is a functionality that might not be as useful for the general computer user, but since this prototype is focusing on power users and being able to quickly work with applications and switching between them, this was seen as a natural addition to meet that need.

Metisse has a way of giving the user an overview of what applications that are currently running on the different desktops. The user can not only see what is running where, he can manage the windows from this overview, moving them quickly between different desktops as well as returning to any of those desktops from that view.

Another source of inspiration is Task Gallery which has an interesting approach to separate running applications as tasks in a 3D world. This concept is combined with a free-roaming approach letting the user walk around among these different tasks and applications.

Andy Cockburn and Bruce McKenzie also gave a great deal of inspiration with the paper on revisiting Data Mountain. One specific concept was the $2\frac{1}{2}$D data visualization. It was showed to be a good approach to place data in order to utilize the human spatial memory, and it was decided to at least give this a try and see if the effect of this would be user-friendly and intuitive.

When working with a lot of tasks, the number of running applications grows. Even if these tasks are divided on different desktops, there might still be a situation when the user needs to spend some time searching for a window. It was then decided to allow a search functionality to search for the windows on the desktop. MS Windows Vista has a similar functionality where the user can search for applications in the start menu instead of wasting time searching through the sometimes vast amount of items in there. A similar functionality was also found in the desktop replacement BumpTop where the user can search among the items on the desktop to find a specific icon or file.

5.1.2 Research on Techniques and Technologies

It was decided to implement a basic rendering engine capable of performing the tasks needed for this kind of prototype. The rendering engine was decided to be constructed with a general design making it possible to use it in other types of 3D applications. It was also decided to make it capable of using different rendering APIs, such as *OpenGL* [25] and *Direct3D* [9], in order to make it more flexible and usable on different platforms. This was done by creating an interface to the rendering system to use from within the application. An interface which can be connected to a rendering API by choice. At this point only OpenGL was decided to be implemented for the prototype.

To handle input it was decided to use an external library called *Object-oriented Input System* (OIS) [24]. This library is open source and handles input from mice, keyboards, and joysticks on MS Windows and Linux platforms.

When constructing a 3D rendering engine, there are some very vital points to take into consideration. How will each object be rendered and how will their orientation in the 3D scene be represented? When the objects are scattered across the scene, how should they be structured in an organized manner? These questions will be answered in the following subchapters.

Object Representation Every object that is rendered in the scene is a set of related vertices. Such an object is often referred to as a mesh in computer graphics. Each vertex in a mesh needs to have at least a coordinate in the 3D space. If a texture is associated with the object a texture coordinate needs to be included in the vertex information. If no texture is used a color can be used instead. A vertex can also have a surface normal in order to be perceived correctly when lightning is used in a scene. For convenience this information about an object's vertices was decided to be stored in files which could be parsed and interpreted by the application runtime. This way you separate content within the program from actual programming code making it easier to control and maintain the system. There are many different file formats available for storing meshes. It was decided to use a simple text based format called ASE (ASCII Scene Export) [3] because of its simplicity and prior experience using it.

Representation of Orientation To represent the orientation of an object in space in a rendering engine there are typically two different approaches used. Either to use matrices or to use quaternions [2, chapter 3.2]. The main benefit of using matrices is the fact that common 3D APIs (Application Programming Interface), such as the ones for Direct3D and OpenGL, use them. When using matrices Euler angles are often used to represent the rotation around the three axes in space, XYZ. Euler angles do have some limitations though. When rotating it is possible to end up with a *Gimbal lock*, which is a term for losing one degree of freedom. Storing the orientation in this manner does also make it harder than it needs to be to interpolate between different orientations.[2, chapter 3]

A solution to both mentioned problems is to use quaternions instead. A quaternion can represent any possible orientation in a 3D space by storing any angle of rotation around any possible axis. Quaternions are relatively easy to use but the mathematics behind is not trivial and is out of scope for this paper. When using quaternions to represent orientation it is fairly simple to get a smooth transition between two different orientations by using spherical linear interpolation. Functionality which can be very useful when one wants to move the camera from one position to another smoothly and change its orientation along the way. [2, chapter 3.3.2] Similar to matrices quaternions are simple to handle. To perform multiple rotations after another you simply multiply them. The obvious drawback using quaternions is the fact that matrices are still needed in the last step when communicating with the rendering API. Fortunately it is simple to convert a quaternion into its corresponding matrix.

Due to their simplicity and flexibility to work with, quaternions were chosen to be used for storing and working with object orientation. Other operations that are used a lot in computer graphics are vector and matrix operations. Instead of trying to implement the math behind these different operations it was decided to use an external library. The choice fell on MathGL++ [18]. MathGL++ is an open source library which has an easy to use API containing the most common functionality needed concerning vectors, matrices, and quaternions.

Scene Graphs and Spatial Data Structures When having multiple objects in a 3D scene a structure that keeps track of them makes the work a lot easier. *Scene graph* is a concept that refers to a data structure in 3D graphics which keeps track of all objects. When you have multiple objects in a scene graph the underlying structure of how this data is stored has huge impact on performance. These structures are usually called *spatial data structures* (SDS). In computer graphics there are several well known and widely used SDSs. Even if they internally work very differently and have different properties, they are all used to group and keep track of objects in space.

For this project it was decided to use a *Bounding Volume Hierarchy* (BVH). A BVH is a tree structure made up of bounding volumes (BV) as nodes. A BV can be seen as a container which encloses other child BVs or objects in a scene. The BVs themselves are generally made up of simple shapes such as boxes or spheres. This makes it possible to discard an object

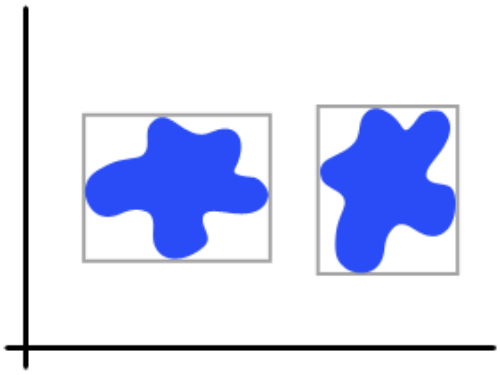
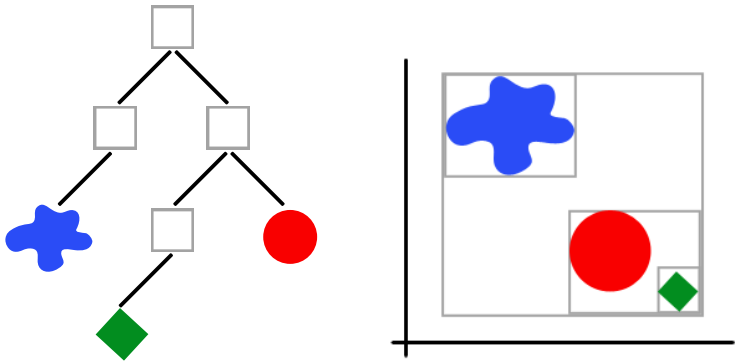


Figure 11: An AABB encloses an object and does always align with the axes of the coordinate system no matter how the object itself is oriented. Here the AABBs in a 2D coordinate system can be seen as gray boxes around the same object oriented differently.



(a) BVH representation seen as tree structure.

(b) BVH representation where nested AABBs are used as BVs.

Figure 12: The concept of using AABBs as BVs in a BVH, showing how the BVs would be nested for a particular BVH tree structure.

if necessary without the need to make any calculations on a 3D model which might be much more complex. When traversing through the tree structure one can skip a lot unnecessary traversing in the tree structure and save a lot of time since every BV is enclosed by its parent BV. This means that any child object in a BV can be ignored when its parent BV is ignored or rejected. This kind of structure can be very powerful if the objects in the scene which belong together, and are located around each other, are grouped and organized in BVs. In Figure 12 one can see how a BVH can be formed of several BVs and objects.

As BV shape it was decided to use *Axis-aligned Bounding Boxes* (AABBs). The main reason for this choice was their simplicity. An AABB is a box which is always aligned with the three axes (X,Y,Z) in the coordinate system no matter of how the contained objects looks or are oriented (see Figure 11). To compute this box for an object one only needs the minimum and maximum coordinates in space for that object. To compute the AABB for a BV this means that you only need to compare the minimum coordinates for all enclosed AABBs with each other and all maximum coordinates with each other to construct a correct AABB. In Figure 12(b) the nesting of the BVs and their AABBs can be seen. Two other common shapes that were discarded were spheres and *Object-oriented Bounding Boxes* (OOBBs). Spheres have the drawback of having to compute distances using a square root which is an expensive operation. OOBBs on the other hand, which are a bounding box that encloses the object but rotates along with the object, give a more accurate result than the AABBs but requires more computations since cal-

culations preferably are performed in the OOBB's local coordinate system.

Since the 3D scenes of this prototype will not be very complex using a BVH as SDS is considered sufficient. A big advantage using BVH compared to some other SDSs is that objects in this structure can be moved around without the need to recompute positions and change in the tree hierarchy. It did also feel like a natural way of dividing the objects used in the scene with a parent-child structure.

By using a SDS to keep all objects in the scene you have greater control over their location. Two areas where this have huge effect are *picking* and *culling* objects before rendering.

Picking To interact with objects in 3D space it was decided to use a 2D mouse cursor instead of any 3D version incorporated into the 3D world. The main reason for this decision was that this prototype is supposed to be used with mouse and keyboard as input devices and that the step from using other WMs should be as small as possible. Since this was an approach that has worked well in computer games for instance, it was seen as a small risk that people would perceive it as unintuitive.

In order to determine what object in the 3D world the mouse is currently pointing at a technique often referred to as ray casting was decided to be used. A ray is vector with a 3D position as origin and normalized 3D vector as direction. The idea is to use a ray that corresponds to where the mouse cursor is pointing in the 3D scene. This ray should then be checked for intersection with objects in the scene.

The choice of BVH as spatial data structure with AABBs as BVs was made with picking in mind. Since a BV encloses all its children a missed BV means that none of its children are hit either. A simple recursive check in the BVH tree structure, where a missed AABB stops the traversing from going deeper, finds all possible intersections. This way a lot of unnecessary intersection tests between the mouse ray and objects in the scene are skipped.

By comparing which objects this ray intersects with you find all objects which the mouse cursor is pointing at. In most cases the object nearest is the one you would like to pick, but since that is not always the case it was important to implement a technique where different hits could be sorted depending on distance.

View Frustum Culling View frustum culling is the process of determining which objects in the scene that are within the view frustum of the camera and to separate them from those objects that are outside.

When rendering this makes it possible to determine which objects that

are outside the view frustum of the camera without going through each object individually. This saves a lot of work when rendering, since these objects will not be visible anyway and they can be discarded. By not passing them on to the geometry or rasterizer stage of the rendering pipeline, the processing time can, depending on the complexity of the scene, be greatly improved.

An interesting paper by Akenine-Möller and Assarson [4] was found for future use. It describes a combination of different methods in order to optimize the view frustum culling for bounding boxes such as those used in this project.

5.1.3 Application structure, Basic Rendering, Input

An object oriented approach was seen as an obvious choice in order to keep the system as easy as possible to design and maintain. By designing in an object oriented manner it is easy to get a good structure of the system since functionality and information can be encapsulated into logical parts called objects. These objects can only be accessed by other objects which are supposed to have access, and information that is only important within each object is not visible from the outside.

As mentioned in the task analysis the prototype's functionality can be divided into different parts: one part which the user interacts with, one part responsible for keeping track of the 3D scene, and another part to communicate with the graphics hardware.

One important design decision that was used throughout the entire system architecture was to use dedicated managers for all main tasks. These managers were decided to be implemented as singletons since that would make them possible to be accessed from other classes without a direct reference to a particular object, but also in order to make sure that there is only one active instance of the class running simultaneously [13, chapter 26.5]. This design makes it possible to divide functionality into logically and conceptually separate parts that do not interfere with each other. It does also give a common public interface for other classes to use, which means less passing of references between objects. This type of design is not unique in any way and is a design that has been used successfully before. The well known open source rendering engine *OGRE* [23] uses this approach for instance and has been a great source of inspiration in this project.

The first step of the implementation was to create a window that could receive input and to render the content of a 3D scene. Since the development environment was MS Windows XP/Vista this was made only for MS Windows. To handle input from all connected input devices a dedicated event handler was created to handle and respond to input received via the

OIS framework. To connect OIS to the application was an easy task, and at this point the event handler did successfully handle input from the mouse, keyboard, as well as game pads.

In order to make the development progress smoother one of the first functionalities that was implemented was a terminal. One part of this was to redirect standard output from the application to a separate console window. This window could then be used to print debug information, and act as a console to the application. The terminal with its separate console window could then be used to print useful information during the development as well as take input to change predetermined settings, change state of the application etc.

When rendering a 3D scene a common approach to update the program is to use listeners which are connected to the frame being rendered. This means that the listeners are called before each frame is rendered as well as after each frame. This approach was chosen in this prototype as well in order to easily and in a structured way be able to respond to input and other events which occur. These listeners are connected to all parts of the program that needs to be updated each frame. By having a timer in the rendering loop the listeners are provided the passed time between each frame in order to respond correctly even for time dependent events.

When rendering a 3D scene the result depends on where the camera is located in space and what direction it is facing. In order to capsulate the settings such as position, direction, and field of view of the rendering, a camera object was created.

5.1.4 Picking Items in Space and Working Dummy OS

Once the basic rendering and input was connected, the implementation of the fundamental functionality could be done. This included a working dummy OS to have items to work upon, a content loader to give these items visual representation in the space as well as basic picking to move these items around.

DummyOS The DummyOS was built to facilitate the running applications and provide information about them to the window manager. If time would allow it, DummyOS would work as a middleware between this window manager and the OS, but since there is no connection between the real underlying OS and the prototype, the DummyOS only keeps a list of fake programs running. Each application initiated by the user is given a unique ID. The applications themselves knows only which application they are, what their title is and what application ID it has been given by the DummyOS.

Content Loading A mesh loader was implemented to be able to open and read text files formatted according to the ASE file format standard. Only the needed functionality of the ASE standard at this point was implemented and the application did now have support for loading meshes and storing positions, normals, and texture coordinates for all vertices in an object. This information was encapsulated in a mesh object and in order to connect a mesh to the 3D scene, functionality in the node objects was added to keep references to meshes. Only meshes that are connected to a node in the scene graph exist in the 3D world and only then are they rendered to the screen. Meshes were decided to not have any position or orientation in the world themselves, instead they are being rendered depending on where and how its parent node is located.

The process of writing this loader went on smoothly and the next step was to enable the possibility to load image files in order to use as textures in the rendering process. There are a number of different image types available. It was decided that support for only one type of image was sufficient at this point. The choice fell on TGA. An image format that has support for transparency and in other ways is easy to work with. At this point it met all the needs of this project. Implementing the loader for this image format was done easily since good support was found online on how to read and interpret the TGA file format [21]. When support for textures was implemented windows could be rendered in the scene by applying textures of application screenshots on a rectangle shaped mesh.

Scene Structure A scene manager was created in order to keep track of all objects in the 3D world. Earlier it had been decided to use a BVH as SDS. This was implemented by creating node objects which had references to both their parent and children. By keeping the root node in the scene manager no additional data structure than this tree of linked nodes was needed for storing the objects in the scene.

In order to have a flexible tree structure where a child node follows its parent when the parent's position or orientation is changed, it was decided to implement the orientations and positions of a node as relative values to the node's parent.

As BVs it had been decided to use AABBs. These were implemented and each node object was provided an AABB. Even if the BVs are abstract containers and should never be seen in the 3D world the functionality to render these boxes was added in order to more easily see that the different parts of the BVH behaved as expected. In order for an AABB to always enclose all of its children's AABBs, functionality was implemented to make

the AABBs update themselves before each frame whenever a child had been updated.

At this point no view frustum culling had been implemented. This meant that all objects in the scene were being processed each frame even if they were not visible. Since the complexity of the scene was not very high this was not a big hit on the performance at the moment but is a problem that should be addressed if optimization is needed.

Picking Raycasting was implemented in order to be able to distinguish what the mouse was currently pointing at. When the ray is shot into the scene, there is a check to see if it collides with any AABBs. If the AABB is hit its children nodes' AABB are checked for collision as well. By testing intersection between the picking ray and the root node recursively all objects that the mouse is pointing at are found.

The ray is easily calculated in terms of unit cube coordinates. The origin of the ray is the screen relative 2D position of the mouse cursor projected onto the near clipping plane of the view frustum. Since the frustum in unit cube coordinates is orthogonal the direction of the ray will always be (0,0,1). The origin and direction of the ray can then easily be converted with some matrix calculations into world coordinates in order to be compared with objects in space. This is done each frame in order to always have a correct ray according to camera movement and orientation.

The algorithm used to calculate intersection between a ray and an AABB is strongly inspired by a version of the slabs method by Akenine-Möller published in Real Time Rendering [2, chapter 13.6.1]. Some other algorithms were briefly investigated but this was chosen for its simplicity to implement and the good result it yielded. As a first version of picking in the prototype only the AABBs of an object were picked and not the objects themselves.

5.2 Milestone 2

The deliverable for this milestone was to have an application where the key concepts of this prototype could be tried out by people with different level of computer experience. The way the work progressed and the decisions about the design that were taken in milestone 1 influenced very much what the deliverable for this milestone was going to involve.

5.2.1 Iterate Work in Milestone 1

At this point the picking functionality was not sufficient since the AABB of an object in most cases is bigger than the actual object. There was a need

for a more accurate solution where one could pick any object exactly if the interaction in the prototype was going to be intuitive and usable. To solve this issue an extra step was added to the picking procedure.

In this second step the mesh of all potential hits received from the AABB comparison must be tested for intersection. For this purpose the ray needs to be checked for intersection with the triangles in that mesh. The algorithm that was implemented to calculate intersection between a ray and a triangle was strongly inspired by an algorithm published in Real Time Rendering [2, chapter 13.7.3]. When this algorithm showed to be very easy to implement and showed good results no need was seen to investigate other algorithms further.

The code behind the rendering was also refactored in order to have a more object oriented approach. Input for the rendering process was not taken directly from OpenGL anymore. Instead the rendering system wrapper communicated with the rendering API, OpenGL in this case, and the rendering system provided the application with information about the rendering state. Some OpenGL specific functionality that was previously used for the origin and orientation of the rendering was now also replaced with functionality implemented in the camera model. By using functionality in the quaternion and matrix libraries the rendering system could now be provided with the correct input regarding position and orientation of the camera without using utility functionality provided in the OpenGL library.

5.2.2 Decide Different Ways of Interaction to Implement

Some simple interaction techniques were implemented in order test which interaction ideas to discard and which to investigate further.

The possibility to move around in the scene was tested in a few different ways. First it was implemented to manually control the viewing position using a keyboard and gamepad in different ways, similar to how you control a character or vehicle in a computer game. The biggest issue with these approaches was the amount of time it took to move to the position of your choice. Another issue was to place the camera in an optimal position with optimal orientation to benefit from the placement of windows and other objects in the 3D world. The other approach that was implemented and tested was to not move the camera manually but instead having some predefined positions and orientations that the user with a command could switch between. This approach addressed both problems with the first approach, but had the obvious drawback of limiting the user in the 3D world.

Since one of the primary goals with this prototype was to implement a system where the user quickly and easily should be able to switch among

running applications the approach of manually moving around freely in the 3D world was discarded. The other approach where a simple command was enough to move the camera into the correct position was seen as superior regarding usability in the case of this prototype.

Since the start of the project it was intended to have 2D windows placed in a 3D world, but exactly how these should be present and how the user was going to be able to interact with them was yet to be decided. With inspiration from various products and papers some different approaches on how the user could manipulate a window using the mouse as input were implemented.

Different approaches that were considered were to to either let the user move the windows freely in all directions or to limit the movement in some way. Thoughts regarding limitation were to let the window follow the current alignment of the camera or letting it follow a visible plane within the 3D world similar to *Data Mountain* [26]. The approach that felt most intuitive was the last one mentioned having the window following a visible plane. This was also the approach that was chosen to be used.

Possibility to rotate a window around its different local axes was also implemented. This functionality was discarded almost directly since it did not add much good, complicating the interaction and making the content in the windows harder recognize.

The last functionality that was implemented and tried out at this point was the ability to scale a window. The intention was that the user could make important windows larger and less important windows smaller. The problem with this approach in 3D was that it felt confusing since a smaller window gave the illusion of being much further away than a large window instead of different sized. At this point it was decided that all windows in the 3D world should be equally sized in order to avoid this confusion.

It was decided to have no windows placed freely in space, but instead to connect each window to a plane. A number of planes should then be placed in the space in a circle. The camera should be placed in the center of this circle rotating around facing these different planes. On each plane a number of windows could be placed. The interaction the user should be able to perform with each window in the 3D world is to move it around along the plane within the plane boundaries, bring it up to 2D to work with, bring it up to 2D to work with maximized, and close it. Once a window was in the work space the window should be able to interact with as normally, i.e. the ordinary 2D controls for moving, scaling, minimizing, maximizing, and closing.

5.2.3 Implement Interaction Functionality

To realize the interaction needed at this point some issues needed to be addressed. The most important parts were functionality in the rendering engine to accomplish visual effects and overlay rendering, managing windows in the 3D world, as well as a framework for navigation and window interaction.

Custom Vertex and Fragment Programs In order to have more flexibility over the visual result when rendering an object, one can load custom vertex and fragment programs into the rendering pipeline. These programs can either be written in low-level assembly that the graphics hardware can interpret, or in a high-level programming language that can be compiled into low-level assembly [11].

The obvious choice in this project was to use a high-level language. With optimizing compilers for the high-level languages and syntax which is easier and faster to both write and understand, low-level assembly was never even an option. Since the prototype at this point used OpenGL to render the 3D graphics the choice of shading language was between the OpenGL specific *OpenGL Shading Language* (GLSL) [12] or the graphics API independent *C for graphics* (Cg) [11]. Since Cg would work even if Direct3D rendering was going to be implemented at a later point this was chosen as the preferred shading language, and was implemented into the rendering engine.

Transparency and Translucent Objects A common issue in computer graphics is to render translucent objects correctly. The reason for this issue is due to how the graphics rendering pipeline works. When an object is rendered the default behavior is to affect a depth buffer which is used to make sure that an object that is placed behind another object is not rendered in front of that object. This generates problems with translucent objects since the result of how a translucent object looks depends on the objects behind, which means that a translucent object requires the objects behind to be rendered before. A solution to this problem is to render all objects back-to-front, or even better to render all opaque objects first and then render all translucent objects back-to-front afterwards.

It might sound simple to sort objects. Often the object's position compared to the camera position is sufficient, but in order to be sorted correctly some special treatment might be necessary since there are a couple of things to take into consideration. If objects have different shapes and orientations it can be hard to determine which object that actually is in front of another. An even harder scenario is if two translucent objects intersect each other. Then there is a need to split at least one of the objects into several parts

where no two parts longer intersect each other and to render these new parts separately in the correct order. [2, chapter 4.5]

In order to be able to control the transparency for an object in the prototype, custom Cg vertex and fragment programs were written which took opacity as input from the application. Since all visible window thumbnails have the same shape and orientation some potential problems previously mentioned could be ignored when sorting the windows. The only issue was the plane the windows were placed on. When tilted the plane was closer to the camera than the windows since the origin of the plane never moved.

The solution to the transparency issues was to create a render queue where the windows were ordered back-to-front and the plane was set to be rendered before the windows.

Overlays 2D objects rendered to the screen in front of the 3D scene are often referred to as overlays. Ability to manage and render overlays was an important part to implement. Different GUI components as well as the actual windows to work with in work mode were intended to be rendered as overlays.

An overlay is a very simple mesh consisting of two triangles put together as a rectangle. These are rendered after the 3D scene has been rendered. By rendering without any depth check the overlays are always rendered in front of everything else, and by rendering them back-to-front no issues with faulty occlusion or transparency occur. To manage creation, destruction, and keeping of all overlays an overlay manager was created.

A special type of overlay was also created in order to be able to write text to the screen. This was needed for e.g. the mouse tooltip and the window search area. In order to render text to the screen a technique referred to as texture mapped text [32] was implemented. The font is stored as a texture where each letter's position corresponds to a certain texture coordinate. Each character that is rendered to the screen is a rectangle consisting of two triangles, which is rendered with the font texture with matching texture coordinates for its particular letter. This technique is fast and versatile. The text can be rendered at any position and orientation, with any color, and with varying size. One drawback is if the letters are scaled too large. In this case over-sampling of the font texture can result in poor quality, but since the size of the text rendered within the application is restricted, this is not an issue.

Desktop Manager It was decided early that, at least as an initial support, the window manager would provide four separate desktops. To accommodate

this, a desktop manager was implemented.

The desktop manager keeps record of the available desktops, and the desktops keeps records of what applications they have on their respective deskspaces, as well as which applications they have visible in work mode. It also keeps track of all the desktop element settings, such as windows sizes, plane dimensions and plane tilt. To help decide when and where to activate different interaction possibilities, it was also decided that the desktop manager to keep track of the available states as well as which state the user has currently activated. There are five different states the user can be in:

- Deskspace state - when interacting with windows on the 3D desktop
- Workmode state - when working with applications in the 2D overlay
- Birdview state - when in overview mode of all the deskspaces simultaneously
- Searchmode state - when searching for running applications, either on a local deskspace or in birdview
- Widgetmode state - when working in the 2D overlay, initiating applications or using gadgets

Widget Manager To start applications from the prototype, it was decided to implement a Widget Manager. From a final product perspective, this area is meant to contain the start menu, as well as different other OS-specific elements as well as support for gadgets such as weather information or hardware monitors. Although this was unlikely to be implemented on the given time frame, the prototype was still designed to allow this extension. In the meantime, it had support for some of the items on the start menu to be initiated.

Window Search To easily find a specific window in a deskspace it had been decided to implement a search function. By using an overlay and connecting input to a text overlay a search field was easily created. First the search interaction was implemented to start a search when the user pressed *Return*. In order to get a better flow it was tested to submit a search after each key input. The result felt more fluent and dynamic so this interaction was kept. Pressing *Escape* was implemented to close the search field and exit the searchmode state.

The idea was to visualize in the 3D world which windows that matched the query by fading out non-matching windows. This was simply done by,

based on the result for each search submit, alter the opacity of the object for the windows in the the active search. The search algorithm at this point was just a simple string comparison, comparing if the input given by the user was contained within the title of each window.

Input handling and Interaction To encapsulate the functionality that should be able to be performed with the mouse a specific class for this purpose was implemented. Based on input from the OIS framework the class was implemented to control what objects the user held his mouse over, clicked on etc. Some key functionality needed for this was to distinguish if the mouse was over an overlay or not, a trivial check that only requires a simple coordinate comparison in 2D. If no overlay was found another check within the 3D world was performed to see if the mouse was placed where it should pick any 3D object instead. If the mouse affected any overlay or interactive object in the 3D world the mouse class was implemented to trigger any corresponding event depending on if a mouse button was clicked, released, or if the cursor was just held over an object.

Window Representation and Window Interaction A running application in the prototype is both supposed to be represented in the 3D world as a thumbnail as well as a 2D application when brought to work mode. This functionality was implemented and encapsulated in a window object. The window object kept a node object with a mesh object for the 3D scene, as well as an overlay object for the 2D representation.

In order to be able to interact with a window it was decided to implement certain zones of the window which should respond to mouse interaction. These zones were called action zones and were made to be either active for the window in the deskspace state or when the window was brought up to 2D in the workmode state. Conceptually these zones were made very similar. When clicked they activated some predefined functionality depending on the action zone type. The different types of action zones that were implemented were: move, close, minimize, maximize, bring up to workmode, and bring up to workmode maximized. These action zones were made relative to the window they belonged concerning both position and size. They were also given icons so the user knew when which action zone was activated. The 2D action zones were made invisible since they were only implemented to mimic the behavior of the most important user interface controls of an application, such as the close button for instance, which was visible on the underlying window texture.

Camera Movement The movement of the camera is very important when moving around in a 3D environment. Especially if the intention is that the user should have a perception of where he is located and what direction he is facing. In order to get a good feeling for the camera rotation between the different deskspaces it was decided to implement smooth transitions. The first step to achieve this was to implement functionality to interpolate between two different orientations. A good solution for this functionality is spherical linear interpolation [2, chapter 3.3.2] which was added to the quaternions library. This functionality was used in combination with a cosine function in order to get a smooth start and stop.

Another detail that was implemented when rotating the camera was to zoom out from a deskspace slightly when starting to rotate away from it, and to zoom in again when reaching the destination deskspace. The purpose with this was to achieve an effect of locking the camera into position when working with a deskpace. The zooming effect was achieved by changing the field-of-view of the camera.

Different durations for switching between deskspaces were tried out and one second seemed to be a good default value to start with. It was then possible to follow the movement at the same time as you did not have to wait for a long time for the switch to finish.

In order to help the user to keep a sense of orientation when the camera is moved around it was decided to use some landmarks. The choice fell on having a distant view enclosing the entire scene. A common approach to address this is to use a skybox, which is a technique where images of the environment are put on the inside of a box.

Six images are needed: one on each side of the box, representing what is seen from a certain point in the six different directions corresponding to the sides of the box. This box is rendered around the camera to give an illusion of a seamless environment in the distance. One way of rendering this box is to make it large enough to contain the entire 3D scene. Another approach is to render the box at the same position as the camera, while not affecting the orientation of the box, before anything else is rendered to the scene. By doing this without affecting the depth buffer the objects in the scene will be rendered "in front" of the box as if the box actually was infinitely far away. This second approach was chosen since it is very flexible and the size of the box does not depend on the objects in the scene, it does only need to be large enough to not interfere with the near clipping plane. For a skybox to give a realistic effect it has to be a quadratic box where all images should have the same origin for the viewpoint and cover a field-of-view of 90 degrees. It is also important that the edges of neighboring images coincide.

5.2.4 Fix Bugs and Usability Before Test With Test Group

To get some structure of the different states of interaction, a simple state machine was implemented into the desktop manager to switch among its five different states.

When working with an application in 2D in workmode state it was decided to *blur* and darken the 3D scene behind in order to make a clear distinction between the windows seen in the 3D world and in the 2D overlays, as well making it clear to the user that the 3D scene is not available in this mode.

To perform effects on the entire 3D scene, such as blur for instance, an approach called post processing was used [20]. An approach that is often used to create effects of different nature [2, chapter 8.8]. The first step was to render the scene to a texture. This texture was then applied to a screen-aligned billboard covering the entire screen. This billboard was then rendered with a customized Cg fragment program which applied blur and darkening depending on input from the prototype. This way the blur and darkening process could be controlled freely and be faded in and out when switching between different interaction states.

In order to have a system that is fast and easy to use for an experienced user, keyboard shortcuts are essential [7, p.490-491]. For this reason a number of keyboard shortcuts were implemented to enable switching between the different interaction states. As default it was decided to use the window key as modifier to trigger all functionality in the prototype. This key is only present on PC keyboards but any other modifier or combination of modifiers could be used instead as well. The main benefit of using this key is that a lot of potential shortcut clashes can be avoided since it is seldom used by other applications.

Functionality was implemented to be triggered when holding down the modifier at the same time as pressing certain keys. By pressing a number on the keyboard the 3D space mode was set to be activated if not already active, and the camera would turn to the deskspace with the corresponding number. The same functionality was decided for the left/right arrow keys, but with the difference that the camera turned to the deskspace left/right to the active one. If space was pressed the prototype was made toggling between the deskspace state and workmode state. By pressing the *s* key the window searchmode state started and *Tab* was used to change mode to the birdview state. By pressing the modifier and releasing it before hitting another key the widget area was brought up.

For mouse driven users a GUI was also planned to be incorporated to handle switching between the different interaction states. Due to lack of time, this functionality was not incorporated before the user tests were performed.

5.3 Milestone 3

Milestone 3 was the last milestone for this prototype. During this milestone it was planned to do some testing with the prototype on a test group, and to take this input to finalize a version of this prototype. Some of the key concepts to test were how the test group would respond to moving around in a 3D environment, how the interaction between 3D and 2D worked, and how the feeling was of moving windows on the planes with different tilts in the 3D scene.

5.3.1 Usability Study and Evaluation

The primary purpose with the usability study was to test the key concepts and core functionality of the intended prototype. The primary target group for this kind of window management system was people who work with several different applications and/or tasks simultaneously, needing a framework for good overview and structure when multitasking among them. Despite of this it was found interesting to see how people with different backgrounds and ways of working with a computer would respond to the system.

A test group with a total of ten people was selected. These people had very different computer experience varying from barely knowing how to check the mailbox, to using the computer daily at work, to them who have been using computers their entire life. Most of the test subjects were MS Windows users, only a couple of them used Mac or Linux as their main OS.

Before testing the prototype a short interview with the test subjects was performed. The intention with this interview was to get an understanding of how the user normally uses different elements of the operating system, and how he usually interacts with e.g. the desktop, open applications and the start menu. After the interview the system was briefly explained to the test subjects. The purpose of the prototype and what differed conceptually and in functionality between systems today was mentioned.

The tests were performed by instructing people to do different tasks and observing them while performing these tasks. The tasks covered more or less all functionality in the prototype. They were told to use the widget area to start programs, manage placement of windows on the deskpace plane with the plane at different angles, switch between different deskspaces, bring up windows to work mode, toggle between work mode and deskspace mode, search among open windows, switch to overview mode, moving windows between different deskspaces in overview mode, switching between overview mode and deskspace mode, as well as using overview mode as an aid to search for a particular application and to switch to its deskpace.

After the test, follow-up questions were asked in order to try to dig deeper into the test subject's opinion about the concepts of the system and how individual parts worked. One of the most important issues was how the concept of working in a 3D environment felt, and how the interaction in this 3D environment was carried out in the prototype. If the spatial memory in 3D helped them remember where a particular application was placed was also of great importance since that was one of the main reasons for the current design of the system. It was also interesting to find out whether they saw any real purpose of a system like this and if they could see themselves using it. These issues and other feedback such as what worked poorly and suggestions for improvement were of interest.

5.3.2 Adapt System According to Usability Study

Some issues when switching between different interaction states needed to be fixed. This was done by redefining the different interaction states within the code and how they were managed. The interaction states were separated into interaction in the 3D world and interaction made in front of this in the 2D overlays. In 3D the user could either be working with a deskspace or be up in overview position. On top of this he can either have no 2D functionality, be using the widget area, searching among open windows, or work with applications.

Most users testing the prototype could adapt to using keyboard shortcuts to change between spaces and do other commands. But they still missed having a clickable interface to achieve these actions. It was therefore decided to implement a GUI to navigate the system. This was connected directly to the desktop manager, since this should be visible at all times, with the only exception of being in widgetmode, which places this in front of all overlays on the screen. Another useful mouse interaction that was incorporated after user suggestion was to switch to the deskspace state if the user was in the workmode state and clicked somewhere in the free blurred out 3D space.

6 Result

The resulting prototype is an application that runs on MS Windows XP/Vista simulating a window manager. No OS is connected to the prototype. Instead a dummy OS keeps and manages dummy instances of running applications. The prototype is implemented to make the user able to navigate in a 3D world placing and accessing running applications on designated areas. Constraints such as limiting how and where windows can be placed are implemented in order to make the interaction as intuitive, easy, and fast to work with as possible. Movement of windows is limited to follow a plane in the 3D environment and all camera movement is controlled by commands meaning that the user does not have to navigate in 3D space himself.

To be able to utilize 3D effectively the prototype uses a 3D rendering engine for the functionality of visualizing and interacting in 3D. This 3D functionality is the backbone of the prototype and is the major part of the application. It was implemented to meet the needs of this prototype having support for the most vital functionality such as loading and rendering of objects and textures from files, supporting custom vertex and fragment programs, supporting post effects, and performing accurate picking. The rendering engine performed well enough to test the prototype but there is room for optimizing various parts; one big feature for instance which was initially intended to be implemented that was skipped due to lack of time was view frustum culling.

6.1 Prototype Functionality

The prototype has four different deskspaces placed symmetrically around the camera position. Switching between them can either be done using the GUI in the lower left corner of the screen or by using keyboard shortcuts. The GUI presented to the user is very sparse. There are a few buttons in the corner that can be used to switch between deskspaces, enter birdview, or start the search among windows. There is also a button to bring up the widget area. The idea is that when the user normally uses the start menu, he has no immediate use of the information from the windows of running applications, and is therefore not disturbed if the widget area takes over the entire screen. So instead of presenting the user with a start menu, the user is redirected to the widgetmode state where the start menu is incorporated.

The different states of the prototype can be divided into states in 3D, and states in 2D in front of the 3D world. There are two different 3D states. The user is either in the deskspace state or in the birdview state.

6.1.1 Deskspace State

The intention with the deskpace state is that the user should be able to organize running windows in the 3D space. If an application is started it is placed in the active deskspace.

By hovering a window, icons showing its action zones are revealed (see Figure 13). The action zones available in this state make it possible for the user to move the window, close the window, or to bring the window to the workmode area. There are two different zones that bring the program to workmode. One zone that opens with a previously stored position and size, and one that opens the window in fullscreen. If a window is brought to the workmode the workmode state is entered.

The windows on a deskspace are scaled to fit four windows next to each other with some space between. This results in windows being big enough to be able to quickly determine the content, without taking up too much space. Since the action zones are limited by the surface of the window, it is essential that the windows are not made too small, or it will be easy to unintentionally click somewhere the user did not intend to.

Windows can be moved around freely within the borders of the deskspace plane using the mouse. In conventional 2D window managers, the physical mouse movement forward and backward corresponds to up and down on the screen. In this state this movement instead corresponds to moving windows along a plane. Although this movement differs from the ways of general window interaction, no users seemed to have any problem with this and adapted to this almost instantly.

6.1.2 Birdview State

To get an overview over the deskspaces and their windows, the user enters the birdview state. In birdview, the user is presented with all four deskspaces from a topdown perspective with the active deskspace upwards. When changing from the deskspace state to the birdview state, the camera is rotated and translated into birdview position. During the camera movement the deskspace planes and their windows are tilted to face the new position of the camera. This way, the user still knows which deskspace he was using before entering this mode and can easily identify the other desktops.

From the birdview viewpoint, the user can easily go to any of the deskspaces, or rearrange the content of the spaces. When switching deskpace the user is taken to the deskspace state.

By moving a window from a deskspace, a ghost copy is made that follows the user's mouse movements. While moving the ghost window between

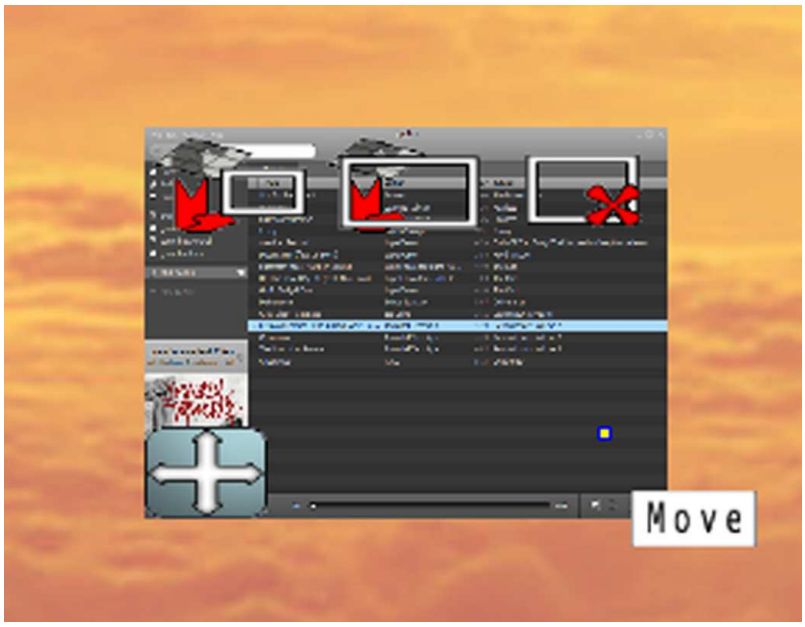


Figure 13: Hovering over a window shows its action zones.

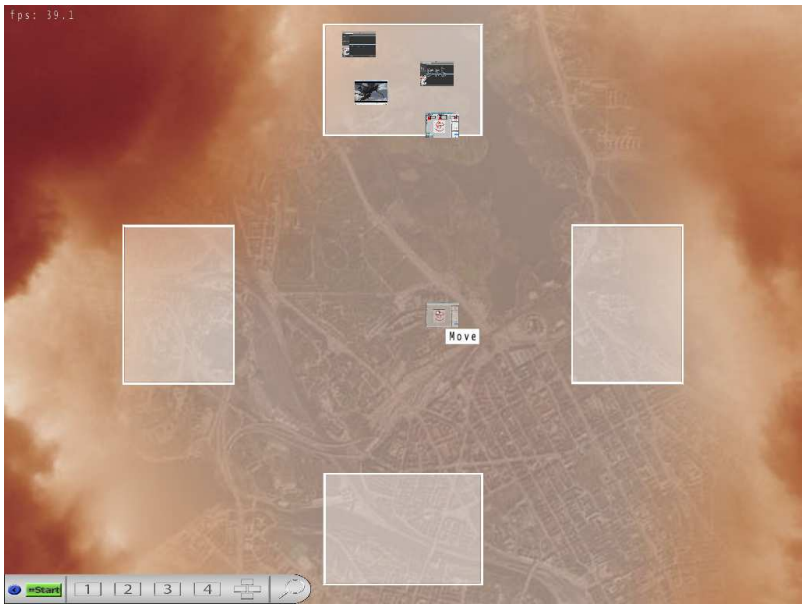


Figure 14: When moving windows between deskspaces, a copy is made to show where it is moving, and the initial window is kept on the current parent space so one knows where it is taken fom.

spaces, the real window follows accordingly along the borders of the current desktop it belongs to. This way, the user can easily see where the window was taken from and it can be put back if it was moved by accident or if the user changes his mind. When the ghost reaches another another deskspace, the window is brought there. See Figure 14 for reference.

6.1.3 Workmode State

The workmode state is where the user works with the applications. This state is in 2D in front of the 3D world. The intention is that working with windows here will be carried out in the same manner as it is done in conventional 2D window managers. If the prototype was connected to a running OS the input from this state could have been redirected to the corresponding running application. Since no OS was connected, the most important functionality is imitated by action zones instead. Functionality available in the prototype is closing, minimizing, and maximizing a window. Since the buttons for these actions are visible on the underlying window texture the action zones are not invisible.

When switching to the workmode state, the 3D scene of the deskspace is blurred out and darkened. This is to emphasize that being in the workmode state working with an application, is not the same state as being in the deskspace. A distinction that is important to make in order be able to easily separate active windows in workmode and inactive windows in the 3D scene behind (see Figure 15).

When being in the workmode state and changing the active desktop, the user is still in in the same state, but is instead presented with the workmode state of that specific deskspace and its active windows. During the camera animation for the deskspace switch, the applications in the current workmode state are faded out in order to let the applications from the other deskspace to be faded in.

6.1.4 Widgetmode State

The widgetmode state is intended to replace the start menu and desktop found in many OSs today. When widgetmode is entered the widget area is put on top of the screen covering the entire screen. A start menu is incorporated into the widget area and from here one can initiate new applications as well as reach other functionality common in a conventional window manager. Besides from the start menu the widget area is intended to be customizable to fit the user's needs by using widgets. Functionality similar to the desktop and the notification area in MS Windows are examples of widgets that can

be incorporated.

Instead of having the desktop cluttered with icons and documents, and having to minimize all windows to gain access to them, one can just open the widget area, and access a smaller file area where the wanted documents or folder shortcuts can be reached. At this point it only works as a dummy screen, with the only functionality to initiate some applications, but the intension with this area is to be fully customizable for the user's needs. See Figure 16 for reference.

6.1.5 Searchmode State

In order to be able to find a window fast if its location was forgotten, a search function was incorporated. By switching to the searchmode state from the deskspace state, a search is initiated among the windows on the current deskspace. If the birdview state was active before the switch the search is global on all deskspaces. When the searchmode state is active a text field in the GUI is visible in the lower right corner of the screen. By typing a search query all windows which's title do not match are faded out. See Figure 17 for reference.

6.2 Prototype Usability

One fear initially was that people were going to experience problems with adjusting to interacting in a 3D environment when being used to 2D. During the user tests nothing indicated that this would be an issue though. No test subject had any real problems to move windows on the deskspaces and they adjusted to different tilts without any problem.

There were various testing with the tilt of the desktop board with the user test group. Using the 0 degree change makes the window placements very similar to any 2D desktop and a lot of the intended purpose of this prototype is lost. Still, one small benefit of this compared to an actual 2D desktop is that you can scale any window image without having to resize the window. In this way one can have a number of windows, different sized or even maximized, and you can still very quickly spot any window you want to work with. By setting the tilt to 90, some users tended to feel that the board was tilting at a decline from the screen, giving the illusion of being more than 90 degrees tilt. This generally got the users to not use the back of the board because of the low visibility. The tilt most appreciated for good visibility and still a lot of room to work with was about 55 degrees. See Figure 18 for references.

Figure 15: When in the workmode state, the 3D world is blurred out so the working windows are emphasized.

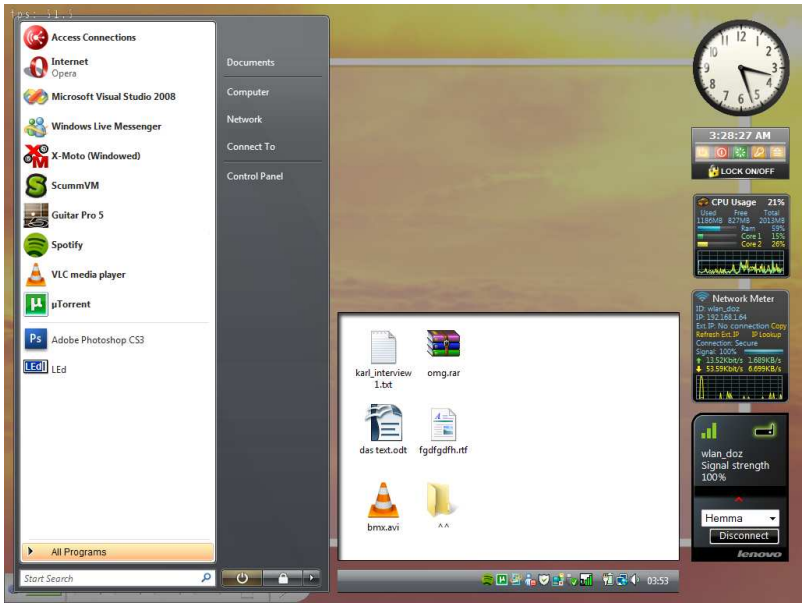
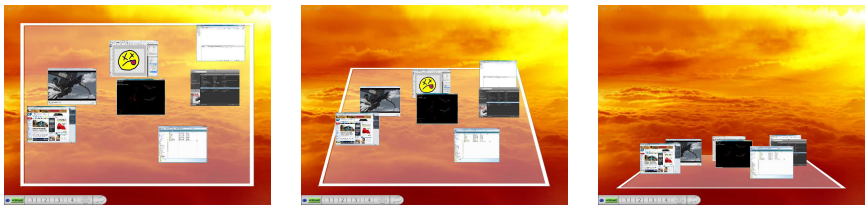


Figure 16: The widgetarea has the start menu fixed and the rest of the area is customizable.

(a) Searching the current deskspace for windows matching the string "photo" while being in the deskspace state.

(b) Searching all deskspaces for windows matching the string "xmoto" while being in the birdview state.

Figure 17: When using the search function, windows that do not fit the search query are faded out.



(a) 0 degrees (b) 55 degrees (c) 90 degrees

Figure 18: Different tilts of the desktop – 0, 55 and 90 degrees.

Neither did anyone express any problem with orientation and navigation in the 3D environment. The navigation was implemented with the intention to avoid making the user feel disoriented by making him able to keep track of where the camera was located and oriented at all times. After understanding the functionality no test subject pointed out any problem to follow the predefined paths the camera took when switching between different locations.

When test subjects were asked to find certain applications they had placed in the prototype there was generally no problem for them to remember where it was. This was also the case for systems used for inspiration testing spatial memory such as Data Mountain.

In the preceding interviews questions were asked about how much and in what way the test subjects used their desktops. Some of the experienced users did not use it at all, but most people used it in some way. Despite of this there was not much resistance to the idea of replacing the traditional desktop with a widget area with similar functionality. To find out if the widget area would work as a replacement in reality, it would need to be implemented and tried out, but on a conceptual level the idea seemed accepted.

The feedback regarding the usefulness of the system was diverse. Test subjects who stated that they did not use a lot of applications simultaneously, tended to have accepted the functionality of their preferred OS without questioning why things worked in a certain way. These people did not see any reason to switch from a system they used, which they thought worked well, to start using another one. A few of the test subjects could see themselves using this kind of system though. In common for these test subjects were that they used multiple applications simultaneously or switched among different tasks frequently. The benefits mentioned were the ability to organize tasks on different deskspaces and finding running applications easier since you have a sense of where you have put it.

7 Discussion

The final prototype has the fundamentals for a working desktop environment, but there are always improvements or different solutions that would greatly improve the value of the product.

We are very pleased with the way the graphics rendering engine turned out. One of the reasons that the rendering engine turned out to be as stable and functional as it is, is probably due to the fact that a big part of the system design was inspired from the open-source rendering engine OGRE, an engine we have been using frequently during our masters program.

For the window manager, this rendering engine does not require much more functionality. One thing that could be implemented, both as "eye candy" as well as an aid for the user orientation is shadows. The most significant drawback from this would probably be some performance drop. It is also possible that the visibility of objects is affected negatively when objects cast shadows on each other.

For the underlying functionality, one improvement would be to add support for Direct3D rendering. Although both OpenGL and Direct3D offers similar functionality and it would make no difference for the visual results of the application, it is still a matter of preference if you want the rendering to be done using OpenGL or Direct3D. Using a different rendering API can affect the performance depending on the system specifications of the running computer.

The biggest improvement for the application at this point would be to actually bind it to an actual operating system. As both authors of this report are MS Windows users, that would be the first operating system to bind to the application. By doing this the system can be used and tried out with completely different conditions, letting test subjects use the system in a real environment and feel how it would affect their work flow. The only feedback received now for instance regarding removing the desktop and incorporate its functionality into the widget area, was based on the test subjects ability to see how it could work since the functionality was not implemented in the prototype. The tests now worked fairly well to test individual parts of the interaction for instance, but to give the test subjects a really good feeling for this kind of system as a whole is hard without a fully working environment.

There has been only a limited research into other OSs. This is because it's hard to gain a good understanding of what a WM can offer unless it is tested. Since we run MS Windows XP / Vista, it is mainly these WM replacements that have been in focused. Both of us has very little experience in the Mac OS family, but there has been some testing of the functionalities using friends' computers, as well as our supervisor, who mainly uses Mac OS,

has shown us some of the functionality it offers. Unix/Linux testing has also been limited, even though it is available at campus. This is mainly due to the fact that these OSs, and their different distributions, are not as hands-on as Mac or MS Windows are. They generally require some scripts and file editing to get new modules working properly. Although these have not been experienced at a personal level, a large number of video captures of different WM solutions are available at different video streaming sites online. One might argue that there has been a lack of pre-studies made, based on these facts. Although we have not been able to try out all the available systems by ourselves, we still consider the information we gathered good enough to be properly evaluated. These demonstrational videos, together with the actual testing of real systems for the MS platform, complements each other enough to give us a good understanding of the WMs available today.

If the system is going to be usable one of the most important improvements would be to add some kind of interface for the users to customize settings. Most parameters and presets that controls the application at this point are either constants that need to be changed in the code or values that require input in a non-user-friendly terminal window to change. Settings that are of interest for the user to customize are e.g. the size on planes and windows, position of planes in relation to the camera, custom wallpapers for the skybox, animation times for the camera movement, tilt on deskspaces, number of available deskspaces, type of action zones, size on action zones etc.

Considering the interaction of the prototype, there are some really interesting concepts in the BumpTop desktop replacement. Instead of being able to stack and group the icons, these actions could be used on the windows on the deskspace. By being able to select many windows at once and either by mouse gestures or by some interface being able to do these actions, it speeds up the reorganizing compared to just moving them one by one with the mouse.Another interesting idea would be to implement collision detection between windows when moving them around. For now the windows just go through each other when they collide but making a window stop when it hits another window could make the interaction more intuitive.

A functionality we think would be usable would be to be able to lock dedicated space on the screen for certain applications, e.g. media players. When another window is activated it is often placed on top of the media application. These programs often have functionality to always be on top of the other windows but this behavior results in them covering important functionality of the underlying window. If an application can be locked to take up a part of the screen, maximizing another window still means that it is only maximized on the remaining area.

The results from the interviews and tests with people did not surprise us very much. Since the intention with the system is to help organizing open programs when working with several applications simultaneously, people who do not use a lot of programs have no need to switch to this kind of system.

8 Conclusion

The main goal with this thesis was to implement a prototype to try out some concepts of using 3D to enhance window and task management. It resulted in a working prototype which uses a 3D world to organize running programs.

The project started with some different ideas on how 3D could be used to organize windows in a way where the user could have more control and overview. Most of the ideas in the initial stage were inspired by flaws experienced in different window managers available. To get more inspiration some research was made. Different products and prototypes available were tried out and research papers describing different areas relevant to the topic were read. Some of these papers and products inspired more than others and the ideas of how the prototype was going to work were reevaluated.

A number of different interaction techniques and concepts were at this point ready to be implemented and tried out. Many of these were discarded at an early stage but a first version of the prototype and its intended functionality was finalized in order to be tested on a group of people with different computer background and experience.

User tests were performed with the primary purpose to find out how people would react and perform using a partially three dimensional and partially two dimensional user interface. The interaction in 3D was intentionally limited in different ways in order to be as intuitive and straight forward as possible when using mouse and keyboard as input devices. No problems were observed for the techniques implemented, neither concerning navigation in the 3D world nor how one interacted with objects in 3D compared to 2D.

The reactions from people were mixed and no strong correlation between test subjects' previous computer experience and appreciation of the functionality of the prototype was found. It was not an uncommon response to question why this functionality was needed. Many of the test subjects were used to the system they used daily and had not reflected on how it could be improved, they had accepted the functionality of their own system and learned to use it and appreciate it. Some positive feedback was received as well from people who liked the ideas and concepts and could see themselves using it.

Feedback from these user tests were taken into consideration and the functionality of the prototype was updated. A lot of further development can still be done though. An interesting next step would be to actually connect the implemented functionality to a running operating system. It would then be possible to evaluate the functionality more thoroughly since people could be using the system in their own working environments and get a feeling of how the system could benefit them during their daily use. This

would probably generate a lot more usable feedback which could be used to develop this system further.

Even if some more functionality and tuning of a lot of details were desired, the prototype fulfills the initial requirements and the final results are seen as satisfying. The implementation process has been instructive and rewarding, and even if some problems have occurred along they way it went on fairly smooth with no major delays compared to the planning. Not all test subjects were positive to this approach of window managing, but the fact that a few were, means that there is some potential in the prototype which would be interesting to investigate further.

9 References

[1] 360desktop official homepage. `http://www.360desktop.com/`. Accessed 2009-05-25. Last updated unknown.

[2] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[3] Ase file format. `http://wiki.beyondunreal.com/Legacy:ASE_File_Format`. Accessed 2009-05-25 Last updated unknown.

[4] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms for bounding boxes. *J. Graph. Tools*, 5(1):9–22, 2000.

[5] Bumptop official homepage. `http://www.bumptop.com/`. Accessed 2009-05-25. Last updated unknown.

[6] Andy Cockburn and Bruce McKenzie. Evaluating the effectiveness of spatial memory in 2d and 3d physical and virtual environments. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 203–210, New York, NY, USA, 2002. ACM.

[7] Alan Cooper and Robert Reimann. *About Face 3.0: The Essentials of Interaction Design*. Wiley & Sons, 3rd revised edition edition, 2007.

[8] Cubedesktop official homepage. `http://www.cubedesktop.com/`. Accessed 2009-05-25. Last updated unknown.

[9] Direct3d api. `http://msdn.microsoft.com/en-us/library/bb173477.aspx`. Accessed 2009-05-25 Last updated unknown.

[10] Niklas Elmqvist. 3dwm: A platform for research and development of three-dimensional user interfaces. Technical Report 2003-04, Department of Computing Science Chalmers University of Technology and Göteborg University, 2003.

[11] Randima Fernando and Mark J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[12] Glsl specification 1.40. `http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.40.05.pdf`. Accessed 2009-05-25 Last updated unknown.

[13] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[14] Kevin Larson, Maarten van Dantzich, Mary Czerwinski, and George Robertson. Text in 3d: some legibility results. In *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*, pages 145–146, New York, NY, USA, 2000. ACM.

[15] Thomas E. LaStrange. swm: An X window manager shell. pages 299–306, Summer 1990.

[16] Litestep official homepage. `http://www.litestep.net/`. Accessed 2009-05-25. Last updated unknown.

[17] Mac osx feature expose. `http://support.apple.com/kb/HT2503`. Accessed 2009-05-28. Last updated unknown.

[18] Mathgl++. `http://mac.softpedia.com/get/Math-Scientific/MathGL-plus-plus.shtml`. Accessed 2009-05-25 Last updated unknown.

[19] Metisse official homepage. `http://www.mandriva.com/archives/en/projects/metisse.html`. Accessed 2009-05-25. Last updated 2008-07-29.

[20] Msdn post processing. `http://msdn.microsoft.com/en-us/library/bb147283(VS.85).aspx`. Accessed 2009-05-25 Last updated unknown.

[21] Nehe tga loader. `http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=33`. Accessed 2009-05-25 Last updated unknown.

[22] Objectdock official homepage. `http://www.stardock.com/products/objectdock/`. Accessed 2009-05-25. Last updated unknown.

[23] Ogre official homepage. `http://www.ogre3d.org`. Accessed 2009-05-25 Last updated unknown.

[24] Ois. `http://sourceforge.net/projects/wgois`. Accessed 2009-05-25 Last updated unknown.

[25] Opengl api. `http://www.opengl.org/sdk/docs/man/`. Accessed 2009-05-25 Last updated unknown.

[26] George Robertson, Mary Czerwinski, Kevin Larson, Daniel C. Robbins, David Thiel, and Maarten van Dantzich. Data mountain: using spatial memory for document management. In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 153–162, New York, NY, USA, 1998. ACM.

[27] George Robertson, Maarten van Dantzich, Daniel Robbins, Mary Czerwinski, Ken Hinckley, Kirsten Risden, David Thiel, and Vadim Gorokhovsky. The task gallery: a 3d window manager. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 494–501, New York, NY, USA, 2000. ACM.

[28] B. Shneiderman. Why not make interfaces better than 3d reality? *Computer Graphics and Applications, IEEE*, 23(6):12–15, Nov.-Dec. 2003.

[29] Greg Smith, Patrick Baudisch, George Robertson, Mary Czerwinski, Brian Meyers, Dan Robbins, Eric Horvitz, and Donna Andrews. Groupbar: The taskbar evolved. In *Proceedings of OZCHI 2003*, pages 34–43, 2003.

[30] Ian Sommerville. *Software Engineering (7th Edition)*. Pearson Addison Wesley, 2004.

[31] Craig Tashman. Windowscape: a task oriented window manager. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 77–80, New York, NY, USA, 2006. ACM.

[32] Texture mapped text. `http://www.opengl.org/resources/features/fontsurvey`. Accessed 2009-05-25 Last updated unknown.

[33] Image source. `http://km.support.apple.com/library/APPLE/APPLECARE_ALLGEOS/HT2503/HT2503_02.png`.

[34] Image source. `http://kodemind.com/strateng/images/desktop2.jpg`.

[35] Image source. `http://image73.webshots.com/173/5/63/49/226956349010379677 3YnGLuP_ph.jpg`.

[36] Image source. `http://www.veriti.net/liens_dyn/img/39128360_slideit.jpg`.

[37] Image source. `http://softmann.com/images/Cube-Desktop-Pro.jpg`.

[38] Image source. http://www.jfedor.org/shots/metisse.jpg.

[39] Image source. http://hcil.cs.umd.edu/trs/2004-29/images/2004-29_img_4.jpg.

A Test person queries

A.1 Questionnare prior to prototype testing

Age:
Gender:
Level of Computer Experience:
Operative System:

- Desktop
  - Do you start programs by icons on the desktop?
  - Do you collect files/folders on the desktop?
  - Can you quickly find what you are looking for on the desktop?
  - Does it matter if you placed those icons?
  - Would you be able to use the computer without the desktop? Why/Why not?
- Window Management
  - Do you often have more than 5 windows open simultaneously?
  - Do you see a need for supporting different task areas, where you can switch between tasks without having to open/close programs?
  - Do you use the mouse or keyboard to switch between windows?
  - Which of these do you prefer: Alt+Tab / Win+Tab / Taskbar ?
- Start Menu
  - Do you use the start menu to start programs?
  - Do you use the start menu to open the computer settings?
  - Do you use the start menu to open folders?
  - Do you use the start menu to open files?
  - Do you use the start menu to do something else?
  - Do you use mouse or keyboard for this?
- Is there any functionality you think is missing?

A.2 Testing the prototype

- Start a program
- Change to another deskspace
- Start a program
- Fetch the program to workmode
- Test interactability with window in workmode
- Change back to previous deskspace while being in workmode
- Go back to deskspace
- Start multiple programs
- Test the search function
- Go to birdview
- Move windows between deskspaces
- Test the search functionality

By doing these actions, all functionality is tested within the prototype, although the users are encouraged to continue try out some things on their own.

A.3 Questionnare after the prototype testing

- Deskspace
  - What did you think about the tilt?
  - What did you think about the window size?
  - What did you think about the plane size?
  - What did you think about the action zones?
  - What did you think about switching deskspaces?
  - What did you think about the search in this mode?
  - Any other ideas?
- Birdview
  - What did you think about the transition from deskspace?
  - What did you think about the distance from the deskspaces?
  - What did you think about the interaction with windows from here?
  - What did you think about the search in this mode?
  - Any other ideas?
- Workmode
  - What did you think about the transition from the deskspace?
  - What did you think about the window interaction?
  - Any other ideas?
- Widget Area
  - What did you think about the idea of having the start menu static and the rest customizable?
  - What did you think about having it on top of the deskspace?
  - Is there any functionality missing?

B System overview - Class Diagram