Interactive Browsing of 3D Scenes in Hypermedia: The Hyper-G 3D Viewer

Diplomarbeit in Technischer Mathematik

Michael Pichler

Technische Universität Graz
Institut für Informationsverarbeitung
und Computergestützte neue Medien (IICM)

Fertigstellung:	6. Oktober 1993
Prüfungsfach:	Computerwissenschaft
Betreuer:	Dipl.-Ing. Keith Andrews
Begutachter:	O. Univ.-Prof. Dr. Dr. h.c. Hermann A. Maurer

Ich versichere, diese Arbeit selbständig verfaßt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Die Diplomarbeit ist in englischer Sprache verfaßt.

Acknowledgements

I would like to thank everybody who has offered suggestions and support during writing of this thesis.
Especially I wish to thank a number of people at the IICM for their support during my work and preparation of this thesis:
Dipl.-Ing. Keith Andrews for his many suggestions and bringing a lot of ideas in discussions with him.
Dr. Frank M. Kappe for his help in finding literature.
Univ.-Prof. Dr. Hermann A. Maurer.
Last but not least I would like to thank all test persons who have "played" with various stages of the implementation for many suggestions and improvements.

Abstract

This thesis discusses methods for interactive navigation through virtual 3D scenes in a hypermedia system. Several methods for manipulating the scene and navigating through it have been developed as well as alternatives for highlighting pickable objects (information links). The implemented 3D viewer has been embedded into the Hyper-G hypmermedia system. Program evaluation was done with heuristic evaluation and usability tests.

Contents

1. Introduction **9**

2. Hypermedia **11**

3. Interactive Browsing of 3D Scenes **15**

3.1. Classification....15
3.2. Requirements....16
3.3. Input Devices....16
3.4. Flip Metaphor....17
3.5. Walk Metaphor....19
3.6. Fly Metaphor....20
3.7. Point of Interest-Movement....20
3.8. Links....21

4. Hyper-G **23**

4.1. Client-Server Architecture....23
4.2. Special Features....24
4.3. Document Types....28
4.4. Applications....29

5. The Hyper-G 3D Viewer **31**

5.1. About the Implementation....31
5.2. User Interface....32
5.3. Navigation....33
5.3.1. Flip Object....33
5.3.2. Walk....35
5.3.3. Fly....36
5.3.4. Fly to....38
5.3.5. Heads-up....38
5.4. Anchors....41
5.4.1. Bounding Cubes....42
5.4.2. Brightness....43
5.4.3. Colour Code....43
5.4.4. Colour Edges....43
5.5. Scene Description....44
5.6. Possible Applications....44

6. Selected Details of the Implementation **45**

6.1. Modularity....45
6.2. Portability....46
6.3. Embedding in Hyper-G....47
6.4. The Scene....48
6.5. 3D Objects....50

6.6. Camera Model 51
6.7. Light Sources 52
6.8. Geometric Objects 52
6.9. Polyhedra 53
6.10. Picking 3D Objects 54
6.11. Materials 57
6.12. Movement 57
6.13. Point of Interest Movement 57
6.14. Anchor Highlighting 59
6.15. Colour Chooser 60

7. Usability Evaluation **61**

7.1. Test Environment 61
7.2. Heuristic Evaluation 62
7.3. Navigation Modes 63
7.3.1. Heuristic Evaluation of Navigation Modes 63
7.3.2. Usability Tests of Navigation Modes 65
7.3.3. "Flip Object" 66
7.3.4. "Walk" 66
7.3.5. "Fly" 67
7.3.6. "Fly to" 68
7.3.7. "Heads-up" 69
7.3.8. Free Choice of Mode 70
7.3.9. Comparison of Modes 71
7.4. Anchor Highlighting 76
7.4.1. Heuristic Evaluation of Anchor Highlighting Modes 76
7.4.2. Usability Tests of Anchor Highlighting Modes 77
7.4.3. General Results 78
7.4.4. "Bounding Cubes" 78
7.4.5. "Brightness" 79
7.4.6. "Colour Code" 80
7.4.7. "Colour Edges" 81
7.4.8. Comparison of Modes 82
7.5. General Results 87

8. Conclusions **89**

App. A) Colour Plates **91**

App. B) GE3D "Graphics Engine 3D" **103**

B.1 Type Definitions 104
B.2 Functions 104
B.2.1 Opening the Graphics Device 104
B.2.2 Display Control 105
B.2.3 Drawing Modes and Attributes 105
B.2.4 The Transformation Matrix Stack 106
B.2.5 Building Transformation Matrices 109
B.2.6 Text 109
B.2.7 Line Primitives 109
B.2.8 Solid Primitives 111

B.2.9 Camera Definition....113
B.2.10 Light Sources....115
B.2.11 Closing the Graphics Device....115

App. C) SDF File Format **117**

C.1 Actor File (ACT)....118
C.2 Position File (POS)....119
C.3 Camera File (CAM)....119
C.4 Light File (LGT)....120
C.5 Material File (MTL)....120
C.6 Object File (OBJ)....121
C.7 Scene Description File (SDF)....122

Bibliography **129**

1. Introduction

As computer performance has increased, **3D programs** have become more and more widespread. The reason is that we live in a three dimensional world and are our senses are naturally accustomed to 3D space.

Currently 3D software is used mainly for two purposes: CAD (computer aided design) and computer animation. *Computer animation* programs create highly realistic pictures, but rendering takes a long time (several minutes per picture) and movements are predetermined. *CAD* programs, used by technical engineers and architects, allow real-time movement and changes to the scene objects, but are in general difficult to use and reserved to specialists.

This thesis discusses a viewer for 3D scenes with the following two properties:

- interactive navigation
- easy interface

Interactive navigation means that drawings are made in real-time and users are able to move the scene or change their view of it. This should give the impression of a "real" three-dimensional scene.

The *interface* has to be simple enough, such that also novice users should have no difficulties navigating through 3D space after a short introduction. On the other hand the program should also appeal to experienced users.

Several methods for *navigation* (movement) through 3D scenes - as easy and natural as possible - are discussed. The wide range of potential 3D models and the diversity of users is thereby considered. The most effective ones were implemented, allowing the user to choose which one to use, depending on the purpose. The term *navigation* can be understood in two ways: more widely taken it means any kind of movement or 3D manipulation, more strictly taken it means changing the view point in 3D space and excludes *object* manipulation. Where it is not clear by context it is specified which term is meant explicitly.

Current hardware needed to draw even relatively simple scenes in real time (at least about 20 to 25 frames per second) is still expensive, but prices are falling and the rendered drawings become more and more realistic.

Hypermedia systems are a way for organising large amounts of heterogeneous data in an easily accessible form. The document types supported in hypermedia usually include text, raster images, 2D drawings, digital audio and video, and sometimes animations.

The defining characteristic of hypermedia systems are the *links* or connections between documents. Such links can lead to documents of other types or to more detailed information, e.g. to a picture illustrating a text. The endpoints of links are called *anchors*. When an anchor is activated (e.g. by a mouse click) the corresponding link is activated.

Hyper-G is a large-scale, distributed hypermedia system currently being developed at the IICM in Graz. Hyper-G also supports many other features, like the clustering of documents, automatic consistency checks, a built-in conferencing system and advanced searching facilities.

This thesis takes the approach of establishing *3D scenes* as document type in the Hyper-G hypermedia system. Given a model of a 3D object the user can examine it from all sides, zoom in for details, and so on. Further, 3D environments can be watched, giving the user a feeling of being present in the scene, a kind of "virtual reality".

For example, in a text document about Graz there might be a link to a raster image of the clock tower (the most famous landmark of Graz). This picture is of high quality, but is still only a flat picture. So there might be an additional link to a 3D model of the clock tower. When the user turns it around and detects that it has a bay on only three out of four corners, this may be a more enlightening experience than reading it in a text.

Objects in the scene may serve as anchors for links to other hypermedia documents. To give a visual cue, which objects in the scene lead to information links, some alternatives for highlighting the anchors were developed.

The range of applications for the 3D viewer varies from technical illustrations (e.g. a model of a car), over models of real buildings (like a famous house) or environments (e.g. a city), up to artificial scenes (virtual realities).

In order to evaluate the various navigation methods and the anchor highlighting methods a number of 3D models were created. First, a *heuristic evaluation* was performed by some students, yielding to some improvements to the 3D viewer.

Particular attention was paid to a *usability test* to get to know which navigation metaphors and link highlights are preferred by novice and expert users and should be offered in a future implementation. Two studies of 32 test subjects were carried out for navigation techniques and anchor highlighting methods. Also some suggestions for further improvements or extensions of the current implementation were made.

2. Hypermedia

The term *hypermedia* is derived from the terms *hypertext* and *multimedia*. One definition of hypermedia can be given as "a system in which documents of various types or *media* are connected with *hyper*-links".

Hyper-text means non-linear text. *Links* exist as connections between documents. It is not necessary to read the documents (often also called nodes) in a predefined linear sequence. Instead when a link is activated the according document is opened. On-line help systems are a common example of hypertext. But also references in lexica, footnotes and literature references (e.g. [Kappe91]) can be seen as links.

Hyper-*media* documents are not restricted to texts. They may also be raster images (e.g. scanned pictures), audio data, 2D object-oriented pictures (e.g. from a drawing editor), video films, animations, and so on.

Hypermedia is influenced by many other research activities, and offers a wide range of research fields. Such related topics are information retrieval, computer aided instructions (CAI), human-computer-interaction, computer graphics, cognitive psychology, computer mediated communication, user interface design, and electronic publishing.

There are two different ways to combine documents of different media: either *frame-based* or *window-oriented*.

Frame-based hypermedia systems use *multimedia documents*, called frames. These are single documents consisting of several parts of information of different types, for example a picture on a sheet of text. The frames are designed and composed by the author including decoration, placement of texts, pictures, and buttons. This full control by the author can be seen as advantage, but is usually a disadvantage since not all authors are also good interface designers, and the frames are difficult to change. Standalone hypermedia systems like HyperCard or Toolbook use this model.

Window-oriented hypermedia systems use *document clusters* to form groups of related documents, but all document types stay separated in their own windows. The author connects the documents which belong together with so-called *cluster links*. This approach is taken by some larger hypermedia systems, like InterMedia, NoteCards, and also by Hyper-G (see colour plate 1 in App. A) as example Hyper-G session).

The window-oriented systems have several advantages. First of all, the hypermedia system is not bound to a particular *user interface* metaphor (like book, travel, library or desktop), because the user can arrange the windows freely. Further the single viewers can be implemented conveniently by different programmers.

Very important is the *extendibility* of the system. A new document type (e.g. 3D scenes) can be added to document clusters by implementing a new kind of *viewer* (or document manager) without changing the rest of the system.

Multilinguality can be supported in window-oriented systems. The same text (or voice) document can be stored in different languages and the system chooses the right one, when the user activates a link to it. Other documents like images are not affected by the selection of the language.

The endpoints of links are called *anchors*. The *source anchor* is the place in a document that has to be clicked to activate a link. Some hypermedia systems also support *destination anchors*. That means that a link can lead to special part of the destination document, for example to a paragraph in an article - when that link is activated the text is scrolled to that paragraph.

A hypermedia system must also provide *searching facilities* for quick information retrieval. This can range from a simple boolean combination of search strings up to a *full text search* including weighting of word occurrences and automatic synonym translation.

Problems of hypermedia can be the "lost in hyper-space" syndrome - users do not know where they are, how to find information known to exist, determining how much information exists about a specific topic and how much of it has already been seen.

A graphical browser can be integrated to show the users their current position. But the documents shown in the map must be selected and arranged carefully, otherwise the map becomes cluttered with links.

An easy way to give an overview are *collections*. Collections group together related documents and/or sub-collections. Thus they form a hierarchy, which can be described as a tree or more general as a directed acyclic graph, when allowing a collection to belong to several other collections.

Current large hypermedia projects are [Kappe93]:

- *WAIS* - Wide Area Information Server. WAIS is based on *full text search* for information retrieval. The information is grouped into a number of databases, and there is one main database which contains keywords of the other databases, such to find the relevant databases in which to search the desired information. There are no links connecting documents.

- *Gopher*. Gopher is based on the *menu* user interface metaphor. It contains no hypertext and is therefore not a real hypermedia system. But it allows key word search and provides a connection of remote databases. It is used for connecting Campus-Wide Information Systems.

- *World Wide Web* - W³. W³ allows the documents to be spread all over the world. The documents are connected with *links*.

- *Hyper-G*. Hyper-G offers a variety of information searching facilities: *links* for connecting documents, *collections* for grouping documents, and *full text search*. It also offers special features like *annotations*, automatic *indexing* and link-generation, a *messaging* and computer conferencing system, animations, *dialogues* and so on, which are discussed in more detail in Chapter 4.

All the above systems are based on the client server model. The servers manage the documents and the clients are document viewers. Hyper-G includes a gateway to information held by the WAIS, Gopher and WorldWideWeb-servers *without* changing the user interface. Likewise the clients of Gopher and WorldWideWeb can access data of Hyper-G over a gateway.

3. Interactive Browsing of 3D Scenes

Navigation in 3D space is complicated by the need to control (at least) six degrees of freedom at the same time: three dimensions for position and three dimensions for orientation. A seventh parameter, the angular field of view could also be considered.

3.1. Classification

3D viewpoint movement can be done basically in two different ways: either moving the *object* (the scene as a whole) or moving the *viewpoint* through the scene. Which one is appropriate depends on the kind of the 3D scene:

1. 3D models of *objects*, which the user wants to move, to look at them from all sides (i.e. turn them around); examples would be a car, an engine or any other technical illustration.

2. 3D models of *worlds* or environments, where the user wishes to feel present in the scene, to walk (or fly) around in the scene; for example a room, a house, or a city.

However, in each of these two groups there are many possibilities for defining a *metaphor*. A metaphor means how the user can understand the behaviour of the interface [Ware90], a kind of an internal model. The user may feel like translating and rotating the object on screen or like walking or flying through the virtual scene.

Another aspect that influences the choice of a movement metaphor is the task it is used for. Characteristic types of viewpoint movements in 3D space are [Mack90]:

- *General movement*. Exploration of the environment, like walking through a building.

- *Targeted movement*. Movement with respect to a specific target point, such as examining a detail in the scene or an object of interest.

- *Specified coordinate movement*. Movement to a precise position and/or orientation, such as specifying that the line of sight should be along the z-axis from point (0, 0, 20) towards the origin.

- *Specified trajectory movement*. Movement along a specified trajectory (curve in 3D space), like specifying a camera movement for an animation.

A technique appropriate for general movement may be inefficient for another type of movement, say targeted movement, and an additional kind of movement had to be offered for that purpose. In general, each metaphor makes efforts to simplify some movement tasks, whereas other actions become rather difficult.

3.2. Requirements

A good viewpoint moving technique should fulfil the following requirements [Mack90, Robert93]:

- easy to use, including easy to learn, to understand, and to remember
- easy prediction of system behaviour in response to the input
- preventing user disorientation or at least an easy way to reverse the movement
- integration with the whole user interface and workspace

Other requirements depend on the purpose of the program. For example, in a CAD application it is important to position objects at an exact coordinate position. Animation software will have to offer a possibility for a smooth camera movement along a specified path. Movement techniques used for viewing 3D scenes, like the Hyper-G 3D viewer, do not have such special requirements.

3.3. Input Devices

A thorough discussion of Input Devices can be found in [Foley90], [Hill90] and [Pimen93]. This section intends to give a short overview over the most important and most common input devices.

The *keyboard*, being the most basic input device, is not suitable for graphic manipulation. The user would have to remember many commands and only the cursor keys offer an association to movement. Nevertheless it can well be used for additional functions and shortcuts for expert users.

- **2D Input Devices**

The input device most often used for computer graphics is a *mouse*. A mouse has two degrees of freedom, horizontal and vertical movement.

The advantages of a mouse are: A mouse is used with almost every computer, most users are very familiar with using a mouse, and novice users learn its usage in a few minutes.

The main disadvantage is that the mouse is a two dimensional input device (although 3D mice have been constructed) and therefore the movement metaphor must build a relationship between 2D input space and 3D virtual space.

Important to mention are also *direct* 2D input devices, which are used directly on the screen. They include the *light pen* and *touch screen*. Although a touch screen may soon be full of finger prints and appears blurred, it can be operated by novice users even more easily than a mouse.

Other 2D input devices are *joy sticks*, *digitising tablets*, and the *trackball*. The trackball is like an upside-down mouse where the ball can be rotated in any direction. Essentially they do not differ from the mouse in the way they can be used as input device for 3D navigation.

- **3D Input Devices**

A 3D input device very similar to the trackball is the *spaceball*. The spaceball consists of a ball that the user can grasp with the hand. Like a trackball the device stays on the table, and only the ball is pushed. In contrast to the trackball the ball is not really rotated, instead the force applied to the ball is measured. Pushing the ball forwards/backwards or left/right and a torsion around its vertical axis gives three degrees of freedom.

Virtual Reality (VR) systems [Pimen93] use a variety of unconventional input devices. A *data glove* has sensors (usually optical sensors at fibre-optic cables) along the fingers to detect the current bending/flexing of the fingers. It can be used to recognise gestures. *Position trackers*, usually integrated in the data glove and the head mounted display (stereo output device) for tracking the *position* and the *orientation* of the hand and the head. So they are actually a 6D input device. *Eye trackers* can track the direction in which the looks - they are not widely in use.

The following chapters will concentrate on using a mouse as input device, because it is the most common (and an inexpensive) input device, and almost every computer user is familiar with using a mouse.

3.4. Flip Metaphor

This metaphor moves ("flips") the *object*, while the viewpoint remains constant.

The following movements are useful:

- translation,
- rotation,
- zooming.

Each of them can be seen in different ways. The first approach is based on the *coordinate axes*. Both translation and rotation are possible along each of the three axes, so the user has seven movement methods: translation along x, y, or z, rotation along x, y, or z, and zooming. Zooming means here scaling, i.e. to enlarge the object as a whole.

The choice of the axis for translation/rotation can be done explicitly be the user (e.g. by pressing a button) or implicitly by taking the one which is nearest to a horizontal or vertical line through the viewport. The latter alternative also allows to choose two axes at the same time for a 2D input device. (Such a coordinate based model is used by Wavefront's Advanced Visualizer [Wave91].)

The other possibility is to use axes parallel to a horizontal and vertical lines in the viewport as axes for translation and rotation. This is a very intuitive method and the user does not need to be aware of coordinate axes at all. The two translations (horizontally, vertically) together with zooming let the user position the object at any point in 3D space. This can be seen as translation in the camera coordinate system (see Section 6.6.).

Also rotation can be done in three directions: around a horizontal axis (rotate up/down), a vertical axis (rotate left/right), and the axis along the line of sight (tilting the object). The last one means to turn the object in a plane parallel to the view plane.

These three rotations can be mapped one-to-one to the possible rotations of a *spaceball*. But also when a 2D input device like a mouse is used, it is possible to simulate a spaceball [Chen88]. The idea is to think of three virtual sliders atop the window as can be seen in Figure 3.1.

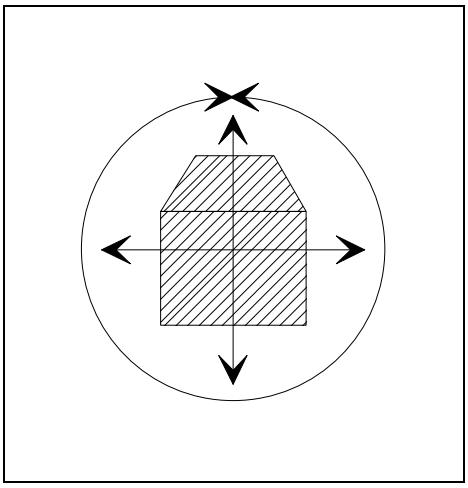

Figure 3.1: Virtual Sliders Atop an Object

Each mouse movement can be mapped onto a virtual sphere. Rotations near the window centre rotate the object left or right, up or down. Rotations near the window corner tilt the object around its centre.

3.5. Walk Metaphor

This metaphor simulates *walking* through 3D space. A similar model would be to drive a car. A movement of the mouse (or other input device) corresponds to a movement of the viewpoint in the scene.

The following movements are possible:

- walking forwards and backwards, and to the left and right side,
- turning the head for looking up and down, and to the left and right.

For more flexible movements this metaphor can be extended by

- movement straight upwards and downwards, like with a lift, or translation (panning).

For the realisation of this metaphor there are still some degrees of freedom. In "walking forwards ..." it has not been specified what *forwards* and backwards means. One possibility is to move the body always parallel to a ground plane (i.e. a plane y = const). This could be appropriate for "planar" scenes, where users rarely wishes to move upwards or downwards - and if so, they can use the *lift*-facility.

The other way is to "follow your nose", i.e. also considering the vertical orientation of the line of sight. This allows also to do a smooth walk along stairs. For movements that should remain on the same level (or height) an additional function for resetting the view horizontally can be offered ("level view").

A complete separation between looking direction and movement direction is not useful - for example when you look to the right side you keep walking in the previous direction unless you also turn your body. This would remove the correlation of the view and the resulting movement and require extra information to be displayed for orientation.

The walk metaphor is appropriate for smaller scenes. The user always has the feeling of controlling the system and any position in 3D space can be easily reached.

Some problems may be the movement over large distances, where the mouse has to be taken off the table, put back and movement has to be resumed, resulting in non continuous movement. Another disadvantage is that movements near objects are rather difficult. For example if the user wishes to look at a detail on a wall he/she will probably walk through the wall.

An improvement can be a non linear *motion function* for making movements over large distances faster, while small movements are fine and well to control.

3.6. Fly Metaphor

One disadvantage of "walking" over large distances is that the mouse has to be moved, taken off the table and moved again several times, which can become tiresome.

In the *fly* metaphor the user feels like sitting in an aeroplane. Input actions are only necessary to change the speed and/or direction of the flight. Even when the flight direction or speed is changed many times, the mouse (or other input device) does not have to be moved outside a small region around its original position.

The following parameters have to be controlled:

- speed: acceleration, deceleration
- direction of flight

In this metaphor the direction of movement will always be the current view direction - without a restriction to a ground plane. It seems plausible to walk through a room looking upwards, remaining at constant height, but the same situation would be unnatural for a flight.

A non-linear, over-proportional *acceleration function* is advantageous. It provides small steps as long the speed is low, allowing easy control of speed for detailed views. When the speed becomes higher, the steps also increase allowing to reach high velocity for movements over large distances.

The fly metaphor is appropriate for large-scale scenes, where it is tiresome to walk. On the other hand *fly* is harder to control than walk and the user may feel to have less control over the system. This can make it inappropriate for smaller scenes.

3.7. Point of Interest-Movement

Point of interest-movement [Robert93, Mack90], or targeted movement as it also can be called, is an approach for combining rapid movement towards a point of interest with controlled movement near the target object. Another advantage, common to the walk metaphor, is that the user need not look at any user interface component, neither real (like menus or displays atop the viewing area) nor virtual (like virtual sliders or scrollers).

First the point of interest (POI) is set by marking any visible point in the scene. The essence of POI-movement is the *logarithmic* moving function. It means in each time step the movement towards the POI is *not* constant (like in the walk metaphor), but proportional to the current distance between the viewpoint and the POI. This means the movement is fast

towards objects and slow near the surface of an object, and it is impossible to fly through faces.

This basic POI movement has the property that the marked point of interest during the movements towards it and away always projects to the same point on the window. Furthermore the target object appears to grow at a constant rate during approaching the viewpoint. This makes it very easy for the user to predict when the object will reach the desired size to look at it in detail.

Often it is also desirable to include an adjustment of the line of sight to be oriented to face the point of interest in a straight direction. For example to look at some text or picture on a wall. This is achieved by *orienting POI movement* [Robert93, Mack90].

The point of interest also gives a surface normal on the face it was hit. Additionally to the movement towards the target point, orienting POI movement includes a lateral movement of the viewpoint parallel to the view plane to put the POI near the centre of the viewport, and a rotation of the point looked at towards the point of interest.

Both movements should also be done logartithmically, but somewhat faster than the plain translation to ensure that the line of sight is near the POI-normal when the desired distance from the object is reached.

Point of interest movement as discussed here provides no means for performing rotations. If a turn is desired an appropriate oriented face must be found to perform the movement, which may be difficult or require an unnatural choice of the POI. Therefore, the point of interest movement is not suitable as a navigation metaphor in itself, but a useful addition to one of the other navigation metaphors, like walk or flip.

3.8. Links

Links, the connections between documents in Hypermedia (see Chapter 2.), can also be used as navigation tool, because a link can also lead to another part of logically the same "world". This is especially useful when the 3D environment is very large and consists of several independent parts.

For example consider a model of a building. When each room is finely modelled, drawing the entire scene would take a long time, resulting in slow, unsatisfying movements. However, its possible to design each room separately and to connect the rooms with links at the doors between them. This allows not only more detailed and realistic models of the rooms at satisfying drawing speed, it also relieves the user from tiresome walks from one room to another.

Another example for this metaphor would be a virtual city, where the user can walk through the streets and click on doors to walk into any building of interest.

4. Hyper-G

Hyper-G (the "G" stands for Graz) is the name of an ambitious hypermedia project currently under development at the Institute for Information Processing and Computer Supported New Media (IICM) at Graz University of Technology. Hyper-G is designed to be a *modular, large, general purpose, networked hypermedia system*. Indeed Hyper-G is going to be the major focus of the institute's work until about 1996.

4.1. Client-Server Architecture

Hyper-G is designed as a *distributed system*. This allows access to large amounts of data by many users simultaneously. The heart of the Hyper-G system, the *core system*, is divided into *clients* and *servers*. They are connected via a high speed local area network (LAN).

A *session manager* controls each Hyper-G session. The session manager starts *document viewers* to display documents and communicates via *ipc*. The session manager also controls link following (the various link types are discussed below). Further it offers on-line help and several navigation aids: an undo/redo function, navigation history, and a map (graph) of the current locality. Furthermore, a collection overview and guided tours (see below) are controlled by the session manager.

All database accesses (for the documents and the links) are done through *database managers*. There are several databases: for the documents themselves, for the links and anchors, for collections and tours. The concurrent server processes can run on different server machines to increase performance.

The core system is not isolated, but connected via a wide are network (WAN) to the rest of the world. This allows system access from remote terminals of different types, but is rather slow (typically 64 kbit/s).

The network also allows access to *remote databases* (e.g. time tables, phone books, satellite data). Thus Hyper-G provides access to millions of documents, which are as far as possible always up-to-date, and could not be kept in a single local system. These documents are like an integral part of the Hyper-G system: they can be *linked* to the local documents and are presented in the same user interface.

The document *viewers* exploit the local intelligence of end-user terminals as far as possible. They retrieve the data over the network from the servers, but then run locally on the terminal. This is much faster than a conventional log-in to a server which is slow and

inefficient: every single keystroke having to be transmitted over a network to the host. In the client-server model most work is done locally (e.g. scrolling a text), only when new data are necessary (e.g. activation of a link) the network and server are accessed.

In general, a number of different terminals are connected to the system. When a Hyper-G session is started, the proper viewer for that terminal type is invoked to take advantage of the terminal capabilities. For example, a VT100-like terminal is able to display text only, in a fixed font, on a window of fixed size, and cursor keys are the only way for scrolling text; images can not be displayed. In an X-window a proportional font can be used, parts of the text can be italic, bold or underlined, and scrolling can be done with a scrollbar using a mouse.

The different viewers can also be used to support different - also unorthodox - input/output devices.

4.2. Special Features

This section can only give a short overview of the most important features of the Hyper-G system. A detailed discussion can be found in [Kappe91].

- **Window Based Model - Clusters**

The Hyper-G system is easy extensible to support new document types or user interfaces. This is possible as Hyper-G is *window-oriented* - in contrast to so-called frame based hypermedia systems, based on multimedia documents (see Chapter 2.).

Each type of document is displayed in a separate window by a *viewer* for that document type independently of the other documents. Documents that belong together are grouped in so-called *document clusters*. This allows Hyper-G to be not tied to a specific *user interface metaphor* for arranging document sequences like book, travel, library or desktop. Instead the user can have the windows arranged as desired and can store the preferences.

When introducing a new viewer (e.g. for 3D scenes) the rest of the system does not have to be changed. The new viewer can be implemented by a person or group who does not have to know details of the implementation of the other viewers. Also each document viewer can use its own file format - there is no need to design a single document standard which covers all types of documents.

- **Identification Levels**

It is possible to use the Hyper-G system *anonymously*. Therefore users need not have the feeling of being watched by a "big brother" and the system can be used from public terminals or in a museum.

Several operations like writing annotations require prior *identification*. Also for storing preferences of users some identification is necessary. Hyper-G supports several levels of identification:

Fully identified users are both known to the system (including name and address) and to other users. They must login with a password.

Semi-identified users are still known to the system, but may choose a (unique) pseudonym and a password. Other users only get to know the pseudonym, but do not know the identity of the user. This mode is suitable for the communication facilities: the user is not recognised by the other users and a misuse of the system is prevented.

Anonymously identified users choose a pseudonym and a password, but their identity is neither known to the system nor to other users. These users are only allowed to make private annotations, which are not visible to other users. In this mode, user monitoring is not possible because the system does not know the user. The semi-anonymous mode still allows the system to remember the preferences of a user between several sessions.

Anonymous users do not need any identification or password. No annotations can be made and user preferences can not be stored.

Identified users may belong to a *group* with certain access rights.

- **Multilinguality**

The user can choose the desired language for the system. Not only the user interface components (like menus, dialogues, help texts) are changed to that language, also the *documents* are provided in the desired language. This is achieved by clusters (see above) of documents with the same contents in different languages.

For example if the user sets his/her language preferences to (1) German and (2) English (in that order) the system will look first for a German text, but will present the English one if no German text is available (or any other if neither one is available).

- **Associative Links**

Historically, most hypermedia systems have used uni-directional links. That means they can be followed only from the source to the destination. In Hyper-G links are *bi-directional*. That means that links can also be followed in the reverse direction (of course the links are still directed).

This backward (also called *associative*) link following can be used to ask which documents contain a link to the current document. For example when reading a text about Graz, the user may ask what other texts to refer it, leading to other texts of interest (for example about the province of Styria or Austria) to which no link may have been provided.

Bi-directional links are also of advantage for link maintenance (see below). When a document is deleted, it can be determined which other documents have a link to it, which then have to be erased.

Associative or bi-directional links require that the links are stored in a separate database and not within the documents.

- **Anchors**

Hyper-G supports both source and destination anchors. All hypermedia systems have source anchors - it is simply the name for the parts of the documents that can be clicked to activate a link to another document or cluster.

A *destination anchor* is a part of the target object which is of interest when following the link. This can be a paragraph in a text, a region of an image or a piece of a sound clip (or others). When the link is activated the user gets a view already targeted on the part of interest. For example a text scrolled to the paragraph of interest or a zoomed view of a map.

A destination anchor need not explicitly be specified. There is always a default anchor covering the whole document.

- **Dynamic Links, Link Maintenance**

To manage the large amount of data it is necessary to include facilities for *automatic link maintenance*. An example is the deletion of all links from and to a document when the document is deleted. Also deletion of outdated documents should be supported by the system. A further step is *automatic link generation*. When a document is edited the system can be asked to propose or automatically generate links to related documents.

Many automatically generated links even do not need to be stored. They can be generated at runtime and are called *dynamic links*. A hypermedia system will include some encyclopaedias and dictionaries. For almost any word in any document (except maybe technical terms) there will be an automatic link into the dictionary explaining that word. It would be confusing to highlight each word and useless to store all these links in a database. Instead the user may click also on non-highlighted words, initiating a dynamic link generation. This is realised by a database query at runtime.

These dynamic links reduce the amount of storage for static links and also reduce the effort involved for link maintenance: In the example above, words deleted from the dictionary are no longer found and words deleted from a document cannot be clicked.

It is also possible to have *associative dynamic links*, which means to find a dynamic link that would point to the current document. For example the user wishes to find all documents about a certain topic. This requires *full text search* in all documents and therefore special data structures (inverted index) are necessary.

- **Collections**

Collections are groups of related documents and/or sub-collections. They form a hierarchical structure. Collections help find information about a specific topic and keep an overview. Especially for novice users, following a hierarchy makes it easier to keep orientation in large amounts of data.

In Hyper-G collections need not be trees: a collection may be a sub-collection of more than one other collection. This results in a directed acyclic graph. For example a collection Biochemistry will belong both to collection Biology and Chemistry, which are sub-collections of Natural Sciences.

As any document in Hyper-G belongs to at least one collection, it is possible to show the users their current position in the collection hierarchy at any time, and allow searching for more general or more specialised documents about a specific topic.

- **Guided Tours and CAI**

A "Guided Tours" is a path (sequence of links) through several documents. The path may or may not be linear - all facilities of hypermedia can be used. The documents are somehow related, but are independent of the collection hierarchy.

Potentially every (identified) user may create a tour (e.g. "my favourite impressions of Australia" or "amazing physical phenomena").

An important application of tours are lessons, or *computer aided instruction* (CAI). Such lessons with links to other lessons and well-designed logical structure are usually authored by experts.

- **Annotations**

Any user (unless anonymous) may become an author by writing *annotations*. Annotations can concern documents and/or links. Either the user writes a new document and links it to existing ones or the user creates a link between existing documents (by specifying source and destination anchor).

The newly created documents may be of any type (there is no special document type annotation). The author of the annotation can be determined by other users. Annotations can be made private, visible to a user group or visible for all.

There are no special editors integrated in Hyper-G. The documents are edited using existing software packages. Converters are available to convert the documents of diverse file formats into the formats supported by Hyper-G.

- **Communication**

One way to communicate in Hyper-G is by annotations (see above). Other users have access to one's own opinion.

The Hyper-G system also includes a build-in messaging system (electronic mail) and a *computer conferencing* system. This computer conferencing system can be used for all kinds of discussions, ordered by a hierarchy of topics. Of course, there can be links to and from conferences to other hypermedia documents. To avoid a misuse of the conferencing system the authors must be at least semi-identified, which does not mean that other users know their real name (see above). Several topics can be only accessible for certain user groups.

4.3. Document Types

A paradigm of hypermedia is to allow access to heterogeneous types of information. Hyper-G is an open system allowing addition of new document types. Currently Viewers are being developed for the following document types [Kappe91]:

- text
- 2D raster images, like digitised pictures
- 2D object oriented drawings, usually used for technical drawings or illustrations
- sound (digital) and speech (synthesised)
- digital movie clips
- maps for orientation
- electronic mail and computer conferencing
- courseware, computer aided lessons (CAI)
- dialogues

... and the *3D viewer* discussed in this thesis.

The incorporation of 3D scenes in hypermedia systems is unusual - to our knowledge Hyper-G is the first hypermedia system to integrate 3D documents. Chapter 5. discusses the Hyper-G 3D viewer in detail.

4.4. Applications

Hyper-G is designed to be usable by general, untrained users as well as expert users. Hyper-G should be adaptive to the various needs of different user groups to allow a wide range of possible applications.

The main application area of Hyper-G will be *information* and *communication systems* for larger institutions, universities and companies.

The current version of Hyper-G forms the basis for the *university information system* University of Technology Graz. It is accessible by all students and members of the university. A wide variety of different users on different hardware are using the system to obtain information about the various institutes and departments, for administrative purposes, as a tool for computer aided instruction, and as a medium for computer supported communication.

The European Space Agency (ESA) will offer the "Genius GDS" service based on Hyper-G from the beginning of 1994. "Genius GDS" stands for "Global Environmental Network Information and User System: Guide and Directory Service" and will give a large number of people access to information about satellites and the obtained data of the earth (mainly images) over Internet. As the information is not hierarchical and contains many potential cross references (user manuals, contact addresses, textual descriptions of image series, satellites, and sensors) a hypermedia system offers many advantages over a conventional database.

A number of smaller *spin-off applications* are surfacing which are mainly pursued by IMMIS - the Institute for Multimedial Information Systems of the Joanneum Research [Kappe93].

One such by-product of Hyper-G was an electronic presentation of Austria at the Expo'92 in Seville, Spain. Here, visitors are presented with a map of Austria ("*AMAPA*"), can zoom in into any region of interest to obtain a detailed map, and can scroll that map arbitrarily. By clicking some of over 1500 icons, they can look at texts, pictures and video clips from all over Austria. All information, including the video clips, is stored in digital form. An improved version of the program was in use at the Expo'93 in Tae Jon, South-Korea.

5. The Hyper-G 3D Viewer

The Hyper-G 3D Viewer enables the integration of 3D scenes as Hypermedia documents and provides interactive navigation facilities. Some objects in the scene can be designated as *anchors*. These objects are highlighted and can be picked to activate a link. Also destination anchors are possible: when arriving at a 3D scene, the camera is positioned as desired.

5.1. About the Implementation

The Hyper-G 3D viewer was implemented using C++ [Strous91] on a Silicon Graphics Personal Iris 4D/35 under the UNIX operating system [Bourne82] with the X 11 window system [Reilly88, Scheif86]. The user interface was built with the InterViews tool kit [Linton89, Linton87], release 3.1.

Three-dimensional graphical output is done with the device independent *GE3D* graphics library - see App. B) - which was implemented in C [Kern88] for GL (Silicon Graphics' native graphics library).

The integration into Hyper-G, concerning communication between viewers and the *session manager*, is discussed in detail in Section 6.3. The communication is done via IPC (inter process communication): the viewer is started as demon process, waiting for activation from the session manager; when active, messages can also be sent in the other direction. The *look and feel* issues of the viewer are separated from the data maintenance of the viewer.

The *InterViews* tool kit offers classes for menus, buttons, labels, boxes, input handlers and so on. They are all abstracted from the underlying window system (like X windows). It is very easy to combine these elements to create a user interface and experiment with different arrangements.

InterViews implements three kinds of UI *"look and feel"*: Motif, SGI Motif, and Open Look. The desired one can be chosen via command line. InterViews also manages the different types of X windows *(visuals)*. Graphic workstations we use support pseudo-colour and true-colour visuals.

Like any other X application, it is possible to define style values (X attributes) outside the program. The program specifies default values for font, cursor colour etc., which can be overridden in the command line or loaded from a configuration file, where the users can specify their preferences. This technique is well supported by InterViews and allows a consistent look and feel for all Hyper-G applications.

5.2. User Interface

The *viewing area* is embedded into the application window (see the colour plates in App. A). The window can be resized without distorting of the view (the aspect ratio of the scene is preserved). At the top there is a menu bar with pull-down menus and below the viewing area there is a row of buttons. The button "show anchors" at the window bottom (see Figure 5.1) is used to switch anchor highlighting on and off (see Section 5.4.).

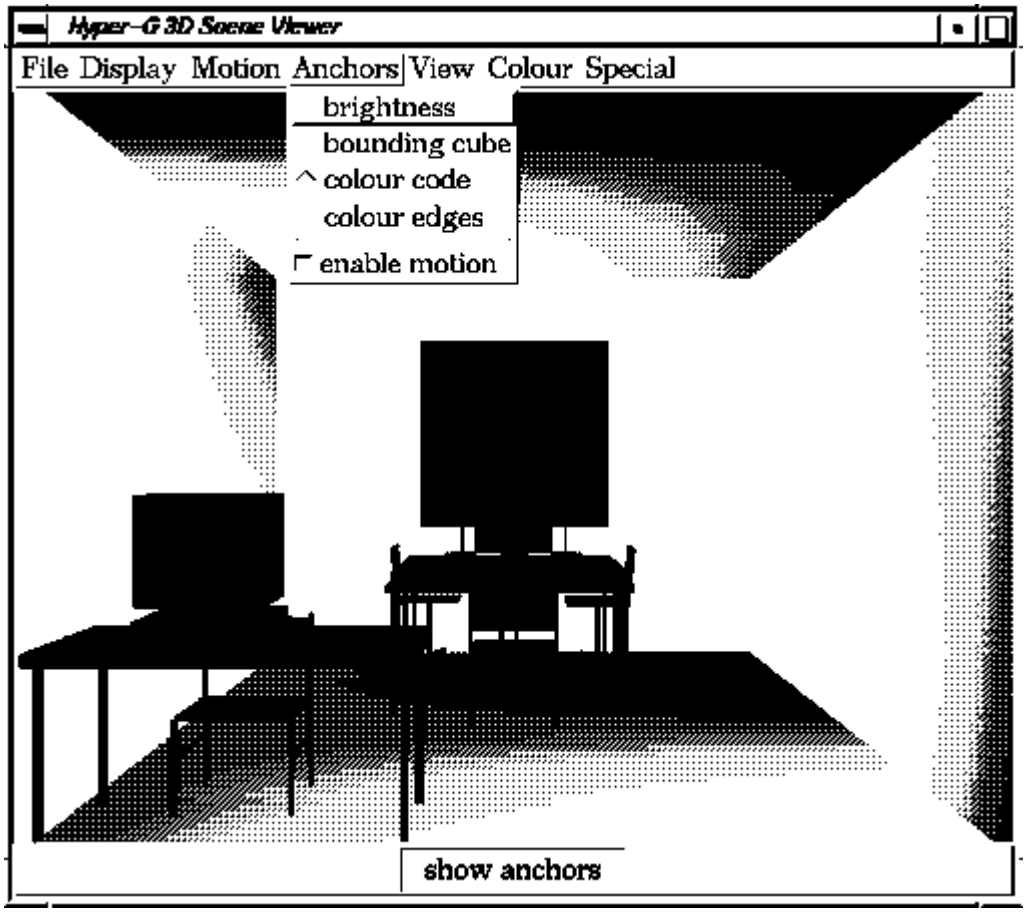

Figure 5.1: Office Scene

Several commands are activated with pull-down menus. The include selection of the display mode, the movement (navigation) metaphor, and the anchor highlighting method. Further the colours for background and UI components can be selected and changed at run time via a colour chooser dialogue.

The pointing device used for navigation input is a three-button mouse as it is available on all computer systems. Some commands need a modifier key on the keyboard to be pressed (SHIFT, CONTROL).

5.3. Navigation

The Hyper-G 3D viewer offers five navigation modes: *flip object*, *walk*, *fly*, *fly to*, and *heads-up*. When *navigation* is more strictly taken it means a movement of the view point, which is achieved by the latter four modes. In contrast, *flip object* is a technique for object movement. The desired motion mode can be selected from a pull down-menu with radio-menu items (see colour plates 2 to 4 in App. A).

Additionally there are two menu items for controlling the *view*:

- *reset view* restores the default viewpoint with which the scene was loaded. Useful when the user totally looses the orientation.

- *level view* rotates the view such that the line of sight is horizontal. This function can be used after looking up and down to continue movement horizontally.

5.3.1. Flip Object

As its name suggests, *flip object* moves the object, while the point of view remains unchanged. The interface can be seen from Figure 5.2 (or colour plate 2 in App. A).

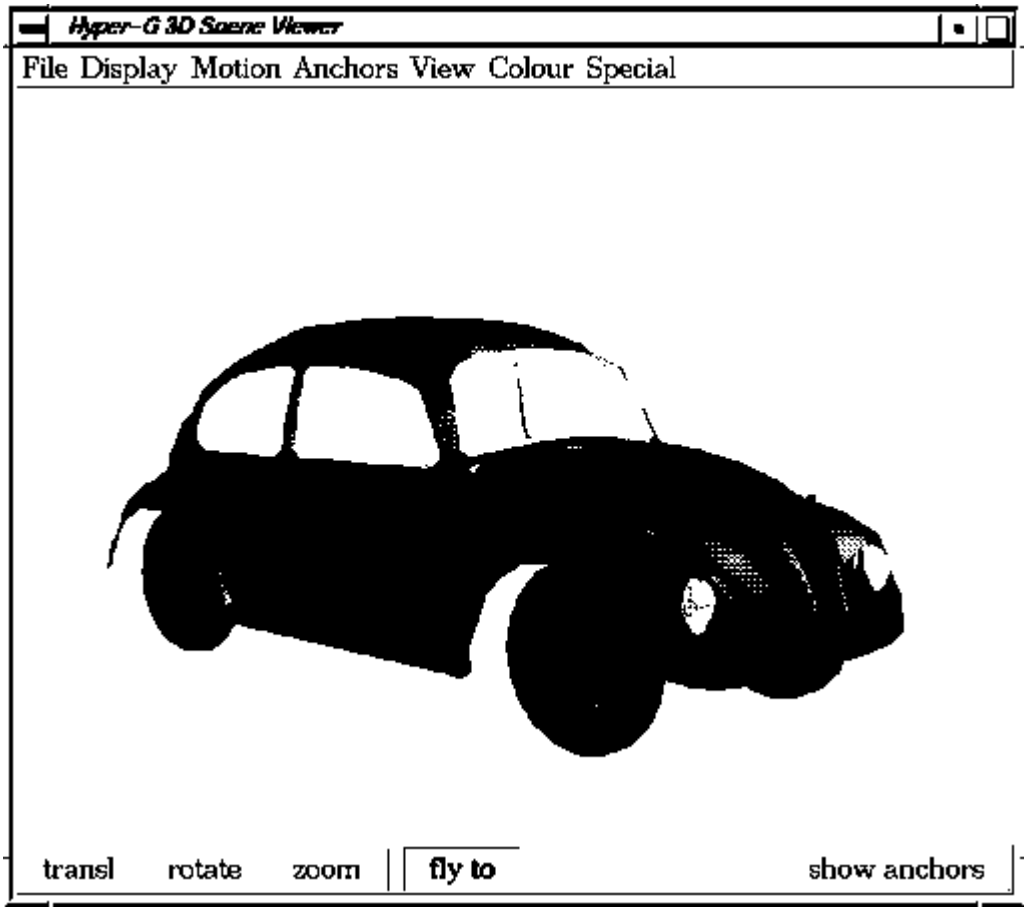

Figure 5.2: Motion Mode *flip object*

Novice users can press one of the buttons at the bottom of the window to select *translate*, *rotate* or *zoom*. The corresponding motion is performed by dragging the left (or any other) mouse button.

Experienced users need not press any of those window buttons. They can use the left mouse button for translation, the middle for rotation, and the right one for zooming. This allows a frequent change of the motion techniques without having to move the mouse outside the scene window. As the mouse buttons are used in the same order as the buttons are arranged in the window, this is easy to remember.

- *translation* means dragging the object with the mouse. It is translated parallel to the view plane and follows the mouse cursor.

- *rotation* is done horizontally and vertically around axes through the centre of the object, diagonal rotations are composites of these two rotations.

- *zooming* - pushing the object away or pulling it towards oneself - is done by dragging the mouse away from or towards oneself (up/down).

An earlier prototype used buttons for translation and rotation along/around the three coordinate axes. It has been changed to this model because is more natural and the user is not interested in the coordinate axes.

Additionally it has been shown useful to be able to get a detailed view of a part of interest in the scene. This is achieved by the *fly to* button in the bottom of the window. When this button is pressed, targeted movement is possible in the same way as in the navigation mode *fly to*. This feature was suggested in the heuristic evaluation and was well accepted by the users (see Sections 7.3.1. and 7.3.3.)

5.3.2. Walk

The motion mode *walk* is a realisation of the walk metaphor. It means that mouse movements result in a "walk" through the scene.

Walk displays no supplementary information, which may sometimes be disturbing, and no extra buttons in the window, requiring additional movements. This mode was designed primarily for experienced users who are willing to learn a few techniques for efficient usage of the system.

All kinds of motion are achieved by dragging the mouse whilst pressing the proper button. To make the assignment of buttons efficient, the left mouse button is assigned the movements used most often, which is to "walk forward" and "turn the head to look left and right". With these two movements all (level) positions can be easily reached. The middle mouse button is used for translations of the view, and the right mouse button for rotations.

Table 5.1 shows the functionality in detail:

dragged mouse button	resulting movement
left mouse button, horizontally	turning the head left and right (horizontal rotation)
left mouse button, vertically	walking forward and backward (z translation)
middle mouse button, horizontally	side-stepping to the left and right (x translation)
middle mouse button, vertically	levitating up and down (y translation)
right mouse button, horizontally/vertically	turning the head in any direction (left-right/up-down)

Table 5.1: Mouse Button Assignment for *walk*

All translations and rotations are understood relative to the current view, i.e. as seen on the window.

To speed up motion over longer distances (both translation and rotation) a non-linear motion function was chosen. Over small distances it still behaves linear, allowing exact movements.

5.3.3. Fly

The motion mode *fly* is a realisation of the fly metaphor. The main advantage of fly is that movements over large distances can be done without tiresome mouse draggings. See Figure 5.3 (or colour plate 3 in App. A) for an illustration of the interface.

As the application runs in a window there needs to be an easy way of starting and stopping motion. This is done by pressing the left mouse button. When flying is active the cursor changes its shape to a (symbolic) aeroplane. When deactivated, the cursor changes back to the default cursor (an arrow) and the mouse can be used for the menus and other application windows.

The *direction* of flight is controlled by the position of the mouse cursor relative to the window centre. Any up/down or left/right mouse motion corresponds to turning into that direction. When the speed is zero, such movements can be used to look around in any direction without translating the view point.

In the middle of the window there is a dead zone. While the mouse cursor is within this area, the direction of flight is straight ahead. The rectangle is the intersection of a vertical strip without horizontal movement and a horizontal strip with horizontal movements only. The latter one makes it easy to keep the same height during a flight. The regions are indicated by lines on the window (see Figure 5.3 or colour plate 3).

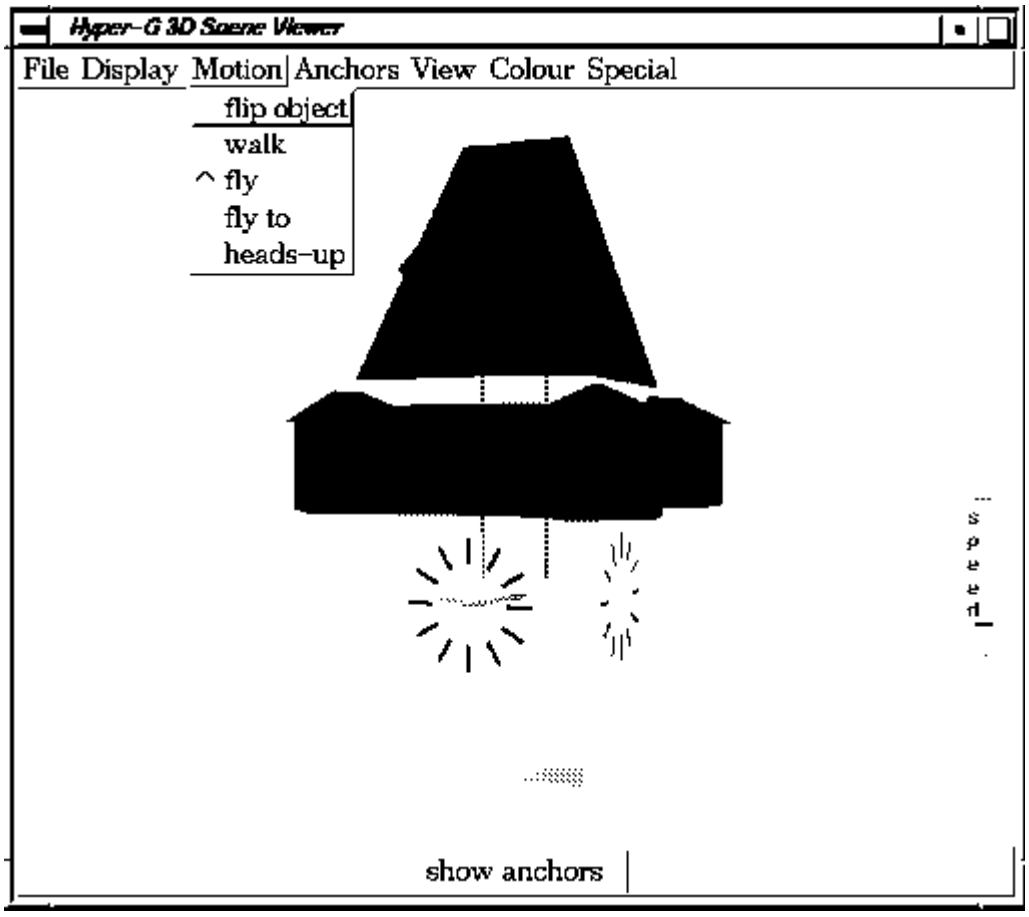

Figure 5.3: Motion Mode *fly*

The *speed* of flight is controlled with the other two mouse buttons. The middle button controls acceleration, the right one deceleration. The current speed is displayed with a slider in the lower right corner. This slider is not used for input, since it would not be possible to change speed whilst simultaneously controlling the direction of flight.

To allow better speed control at low levels, a non-linear *acceleration function* is used. This also allows control of larger range of speeds, because at high speed it is not necessary to have fine control. The transformation of the values is hidden from the user - the speed bar shows a constant change of speed per acceleration step (mouse click) which is not the truth but subjectively corresponds better to the perceived motion.

Similarly, a non-linear function is used for motion control. When moving the cursor out of the dead zone, motion starts slowly and becomes faster the farther the cursor is from the centre.

Additionally speed is reset to zero each time flight is stopped (with the left mouse button), meaning that each flight has to be started by accelerating again. This has shown useful to allow a controlled start of each flight and it is more convenient than reducing the speed manually or using another command.

5.3.4. Fly to

fly to is the implementation of point-of-interest motion as discussed in Section 3.7. and [Mack90].

Zooming with *fly to* is both rapid and controlled. That means that motion is fast when the target object is far away and controlled as the viewpoint approaches the object slowly without flying through or into the object.

Whenever the user is in *fly to* mode (either as separate motion mode or temporarily through another mode) the cursor changes to a crosshair with a circle around it. This requests the user to specify a point of interest in the scene.

The point of interest or *target point* is selected by clicking the left mouse button. For feedback the target point is then marked with circles and a cross. These circles lie in the same plane as the surface on which the target point was selected. By clicking the left mouse button again, the target point can be repositioned at any time.

Movement towards the target point is achieved by pressing the middle mouse button; it can be held down as long as movement should continue. Movement away from the target point is achieved by pressing the right mouse button.

The 3D Viewer allows to press hold the SHIFT key whilst holding the mouse button to move towards the target point. Then the line of sight is adjusted to look straight onto the target point (more precisely: onto the plane of the hit face) and translated slightly to move towards the midpoint of the window. This gives a good view when watching at a detail of the scene.

The usability tests have shown that this adjustment is unnecessary and often even disturbing when *fly to* is used for general motion (see Section 7.3.6.). Thereafter the implementation was changed such that a pure translation is done by default. It is characteristic for this translation that the point of interest always projects to the same point on the window during the movement.

An adjustment of the view direction without changing the distance from the target point can be achieved by pressing the CONTROL-key when holding the middle or right mouse button. The viewpoint is panned to face the point of interest in a straight line at the middle of the window with the middle mouse button and away back with the right mouse button.

5.3.5. Heads-up

Heads-up is also implemented based on the walk metaphor. In contrast to *walk*, it overlays icons on the window (like a heads-up display). These icons should minimise the memory load for the user. Each icon is activated by dragging the mouse out of the icon. A line follows the dragged mouse to indicate which icon is active and the amount of movement.

An implementation of the heads-up user interface is integrated into the *3D rooms* environment of XEROX' *Information Visualizer* [Robert93, Card91].

As it can be seen from Figure 5.4 (or colour plate 4 in App. A) there are four icons:

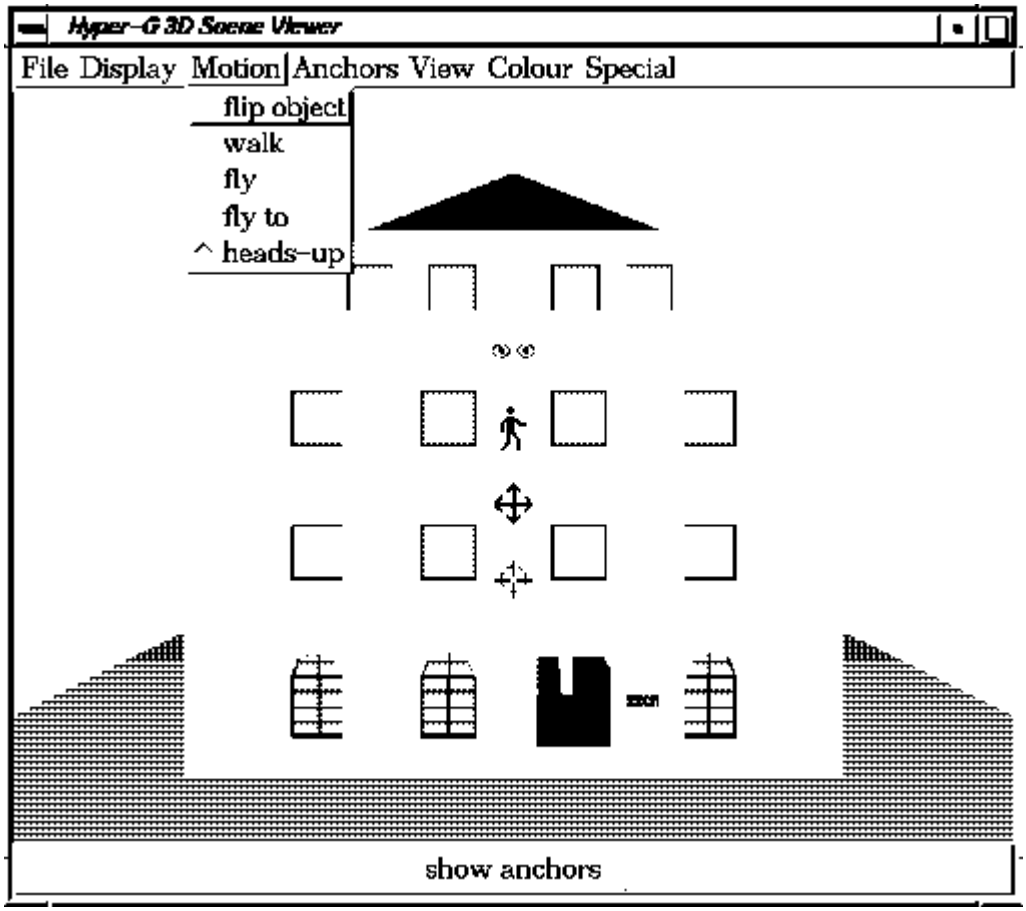

Figure 5.4: Motion Mode *heads-up*

Table 5.2 lists the movements associated with the icons. The functionality was designed in consistency with *walk*. Therefore users can take advantage of experience obtained with *heads-up* when they feel they can do it without information display and want to change to mode *walk*.

	eyes for looking around (rotation around the current viewpoint)
	body for walking (forward/backward) and steering (left/right)
	translation arrows (translation upwards and downwards)
	crosshair symbol (to activate *fly to*)

Table 5.2: Icon Functionality in Mode *heads-up*.

When clicking the last icon motion *fly to* is activated and the icons disappear. Instead the cursor becomes a crosshair symbol (similar to the icon) it is possible to mark a target point. As discussed in the section above the middle and right mouse button are used for zooming in and out. By pressing the left button again it is possible to reposition the target point. *Fly to* is exited with a double-click and the icons reappear.

This motion mode is very natural to use, easy to learn and to remember. Some disadvantages are that the icons may disturb experienced users or may be not clearly recognisable.

5.4. Anchors

Anchors are the endpoints of information links in hypermedia. *Source anchors* are the selectable parts of a document which activate a link. In a 3D scene, the geometric objects populating the scene (see Section 6.4.) can be used as anchors.

A *destination anchor* is the position to which the destination document has to be "scrolled" when a link to it has been activated. Destinations in 3D space can be described with a camera position (view point) and lookat point (see Section 6.6.).

This chapter concentrates on the highlighting and activation of source anchors. Anchor *highlighting* can be turned on and off with a button at the bottom of the window (see colour plates in App. A).

The *activation* of anchors should be possible at any time, also mixed with navigation without having to change mode. This was achieved by using a SHIFT-left mouse click during navigation for link activation. When the anchors are highlighted, there is also the possibility to click anchors normally, but then navigation is no longer possible. The latter alternative was found inferior in the heuristic evaluation (see Section 7.4.1.). Both possibilities are offered and can be switched with the toggle menu item *enable motion*, which is on by default.

When the user clicks in a point on the window, a ray is cast into the scene (see Section 6.10.) to determine which object of the scene was hit.

In order to highlight anchor objects (i.e. the pickable objects), several methods were implemented. The highlighting methods have two contrary requirements:

- it should be easy to recognise which objects are anchors and which not,
- as much realism of the scene as possible should be preserved.

Some alternatives were rejected after preliminary consideration:

Providing a cue, whether the object where the mouse currently points to is an anchor or not. Such a cue could be an audio signal, flashing the object, or a change of the mouse cursor. However users wishing to look for all available anchors have to scan the whole scene by sweeping the mouse, which is rather tedious. Besides a possible performance problem - if the cursor is moved to fast over small objects the cue may come too late or not at all - such cues would disturb the navigation facilities.

Another possibility would be to give a cue only when the user asks for it, for example when clicking an object, some change occurs if it is an anchor. In this way it would be even harder to find out which objects of the scene are anchors.

In some cases it may not be necessary to highlight the anchors at all, for example if almost every object in the scene is actually an anchor - or all objects the user might assume lead to further information actually do.

When looking at an object the eye receives several parameters simultaneously:

- the colour (hue) of the face
- the brightness of the face
- the edge colour
- the object silhouette

To highlight objects some parameters can be modified and the others are left unchanged to let the scene look similar to the original one. Another possibility is to introduce new scene components like a bounding cube around anchor objects or arrows pointing to anchors to distinguish them from other objects.

Four strategies for highlighting anchors were implemented: *bounding cube*, *brightness*, *colour faces* and *colour edges*. The desired one can be selected from the pull down menu *anchors*. As the modes are hard to compare with black and white figures the reader is referred to App. A) for illustrations (Figures 5 to 8).

5.4.1. Bounding Cubes

In this mode (see colour plate 6) all objects in the scene are drawn unchanged, and only an additional bounding cube is drawn around each anchor object. The wire cube is by default yellow-red dashed, but the user can change the colour as wished.

Bounding cubes do not change the properties of the objects, do not disturb, and are easy to recognise.

The disadvantage of bounding cubes are that they are misleading, what should the user *pick*: the whole bounding cube or just the object. If the user is allowed to pick the bounding cube (what seems natural and is easy to implement) it can happen that one cube encloses a smaller one, which is then not pickable any longer, e.g. the headlights of a car. To avoid a depth ordered list of hit objects the current implementation also relies on picking the objects themselves like in all other modes. Further this mode is optically unappealing and may be discarded in future.

5.4.2. Brightness

As the name suggests, *brightness* modifies the brightness of objects. Anchors are drawn brighter than the other objects by modifying the L component in the HLS colour system [Foley90]. The lightness L can be transformed to [0.6, 1] for anchors and to [0, 0.4] for non-anchors, for example.

See colour plate 5 for an illustration of this highlighting mode.

This mode is rather promising in theory, but rather poor in practice. To make a clear difference between anchors and non-anchors all anchors have to be very bright and all non-anchors very dark. The resulting picture no longer looks natural as the human eye is highly sensitive to changes in brightness.

5.4.3. Colour Code

This mode assigns a certain highlighting colour to all anchors, whereas non-anchors are drawn in a shade of grey. The brightness of non-anchor objects is unchanged, only for anchors it is transformed into an interval like [0.2, 1], so that black objects can also be recognised as anchors. A red-brown colour was found to be appealing, but the user can change it as preferred.

As opposed to *brightness*, *colour code* seems at first thought to be rather poor, but turns out to be very satisfying when implemented. This fact was verified in the usability tests (see Chapter 7.4.). The (almost) same brightness lets the scene look like before and the different colours ensure that anchors are easily distinguished from other objects.

Compare colour plates 5 and 7 to see the different impressions of *brightness* and *colour faces*.

5.4.4. Colour Edges

Mode *colour edges* draws all objects in their natural colour, and additionally draws the edges of anchors in a fixed colour. The edges are drawn in yellow by default since it is a bright, attractive colour. Of course the user can configure another colour.

Colour plate 8 shows an example for highlighting mode *colour edges*.

This mode seems to be the best approach. The objects in the scene are not changed very much and it is clear which objects are anchors and which not. It is also clear where to pick - the object itself. However, the usability tests (see Chapter 7.4.) have shown otherwise: small objects are difficult to see and curved objects with many faces are unappealing due to the many wired edges.

5.5. Scene Description

The 3D scenes are modelled with the *Wavefront Advanced Visualizer* [Wave91] (see also App. C). For the purpose of modelling a static 3D scene the programs *Model*, *Property*, and *Preview* are used.

- *Model* is used to edit each 3D object, by modelling it with polygonal faces.
- *Property* is used to edit material definitions (colours) and light sources.
- All objects are composed to a 3D scene with program *Preview*, which means to specify the position of all objects in the scene.

The individual files generated by the Wavefront programs are concatenated into a single scene description file, called SDF (see Section C.7), containing all information of a 3D scene. Other file formats for 3D data, like PHIGS [PHIGS89] or the Renderman Image Bytestream interface [Upstill] may be supported in future.

5.6. Possible Applications

Some examples for 3D scenes as hypermedia documents have been already modelled: an office and a model of the IICM institute (see colour plates in App. A). Consider such an *office*. Clicking on a chair shows an autobiographical statement, on the desk shows a description of the employee's work and a raster image, on the computer plays a video film, on a book shows an overview of the publications, ... Finally, clicking on the door leads to the next office.

Such environments of buildings and rooms can be used for virtual museums or touristic guides. For example, a touristic guide to Graz may contain information about culture, attractions, landmarks, accommodation, cost of living, science, economy, and so on. Many themes have a nice completion in a 3D model of an environment, for example the clock tower, city hall, opera house, and university institutes.

Another possibility is to make use of a 3D object as an illustration. For example, take the description of a *car* in an automobile manual. The first node is a 3D model of the whole car. Clicking on the bonnet would lead to another scene, with the bonnet open, exposing the engine. Clicking on several parts leads to a detailed 3D illustration and a textual description of that part, and maybe also a sound clip.

Finally, large collections of hypermedia documents can be represented as a 3D scene using the *library* metaphor. The scene could consist of one or more rooms, each containing shelves of books about a certain topic, a card index, a librarian, and so forth. The user can walk through the rooms and look for the desired book, which can be opened with a click. Clicking on the card index can bring up a search dialogue. A novice user can click on the librarian for a guided tour.

6. Selected Details of the Implementation

6.1. Modularity

The task of the 3D viewer can be briefly summarised in four points:

- Reading the data, building the data structure to store the 3D scene.
- Displaying the 3D scene.
- Handling the user interface, including menus and responding to input actions like mouse movements.
- Communication with the session manager, to display the appropriate scene and to report the activation of links.

For better program maintenance and to increase the program's portability, the following components were clearly separated:

- Drawing is done with the device independent interface of *GE3D* (see App. B).
- The code managing the *scene* is independent of any window system and user interface.
- The connection between *window* management and GE3D is performed by a module called GE-context.
- The communication protocol with the *Hyper-G* session manager is described in Section 6.3. Embedding in Hyper-G

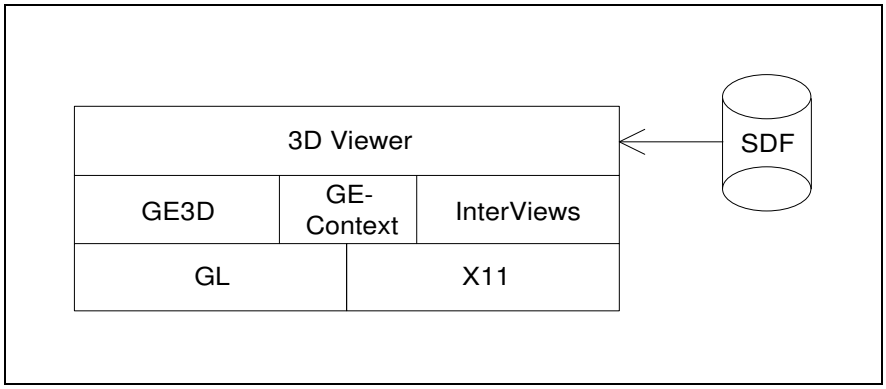

Figure 6.1: Program Components

Figure 6.1 reflects the current implementation: GE3D (see App. B) is based on Silicon Graphics' Graphics Library GL, and the InterViews library is used as interface to the X11 window system. The file format SDF used for description of 3D scenes is discussed in App. C).

6.2. Portability

What has to be done to port the Hyper-G 3D viewer to another platform?

As can be seen from Figure 6.1, it depends on which of the system components is substituted by another one.

- **3D graphic output:** Port GE3D to another graphics library.
- **Window system:** Port InterViews from the X-window system to another window system.
- **Scene description:** Convert files to the SDF format, or extend viewer to read in another file format.

With the appearance of OpenGL [OpenGL93] the current *GE3D* implementation for GL can be adapted to support Open GL, which will be a standardised low level 3D graphics interface offered by many vendors - as opposed to the current hardware dependent interfaces (such as GL from Silicon Graphics and Starbase from HP).

For interactive (real-time) drawing of 3D scenes, special graphics hardware is necessary to do hidden surface elimination and smooth shading. Without such hardware it would only possible to animate wire frames and render static pictures in software.

When another window system than X-windows is used there are two possibilities. The first is to port InterViews to the other window system. Window-dependent InterViews source code is already separated from other (window independent) parts. An InterViews port to Windows NT is already available.

The other approach is to port the user interface code of the 3D viewer currently based on InterViews directly to another environment. This would also include event processing and input handling, which could be quite substantial work, but may be potentially faster in execution.

When changing either graphic output or the window system, the program part providing a link between them - here called GE-context - must be adapted. This module is responsible for opening, placing and resizing the window for graphic output within the application window.

To display 3D scenes stored in other file formats the files should be converted into Wavefront's file formats (which builds the basis for the SDF file format - see App. C). In future the Hyper-G 3D viewer may be extended to support other common file formats (ASCII or binary) for 3D data, like the PHIGS [PHIGS89] standard or the Renderman Image Bytestream interface [Upstill].

6.3. Embedding in Hyper-G

All document *viewers* of Hyper-G are derived from a base class *HgViewer*. It describes the functions the *session manager* can call on any viewer:

- loading a document and telling the source anchors
- displaying a document, optionally browsing to a destination anchor
- terminate the viewer (e.g. when the whole Hyper-G session is closed)

The following excerpt of *class Hg3dViewer* shows the public interface of the Hyper-G 3D viewer (the interface currently may be subject to small changes or extensions):

```
class Hg3dViewer: public HgViewer
{
 public:
   Hg3dViewer (HgViewerManager*);
   virtual ~Hg3dViewer ();

   virtual void load (              // load document
     const char* doc,                 //   document info
     const char* anchors,               //   source anchors
     char* info = 0                 //   response to v.manager
   );

   virtual int port () const;         // return document port

   virtual void browse (              // browse current document
     const char* dest = 0             //   opt. destination anchor
   ) = 0;
```

```
    virtual void terminate ();          // terminate viewer

}; // class Hg3dViewer
```

The communication in the other direction is also encapsulated in classes. Each viewer when created gets a pointer to the *HgViewerMangager* (the session manager as far as concerned with the viewers) to report activation of source anchors, possible errors and the termination of the viewer. Communication between viewers and session manager is done via IPC (inter-process communication) and is bi-directional.

When the viewer demon is started it looks for a free port number for the document port, which the session manager will ask for with function *port()*. For loading a document the session manager calls function *load()* and provides a string with information about the current document (id number) and a string with the source anchors. The viewer can now read the document over the document port. Argument info can be used as a feedback to the viewer manager.

To display the current document the viewer manager calls function *browse()*. An optional destination anchor specifies the destination in the document - a camera position and lookat point in case of 3D scenes. When an anchor is activated, the viewer calls function *follow_link()* of the viewer manager, which is responsible to initiate display of the corresponding document (or all documents of a cluster).

All *look and feel* components of the viewer are implemented in a derived class. In the case of the 3D viewer it is called *IV3Dviewer* as it was implemented with the InterViews tool kit.

6.4. The Scene

All data related to a 3D scene is managed by class *Scene*. The most important data kept by the scene are four lists:

- a list of cameras, and a pointer to the active camera
- a list of light sources
- a list of (geometric, draw able) objects
- a list of material definitions

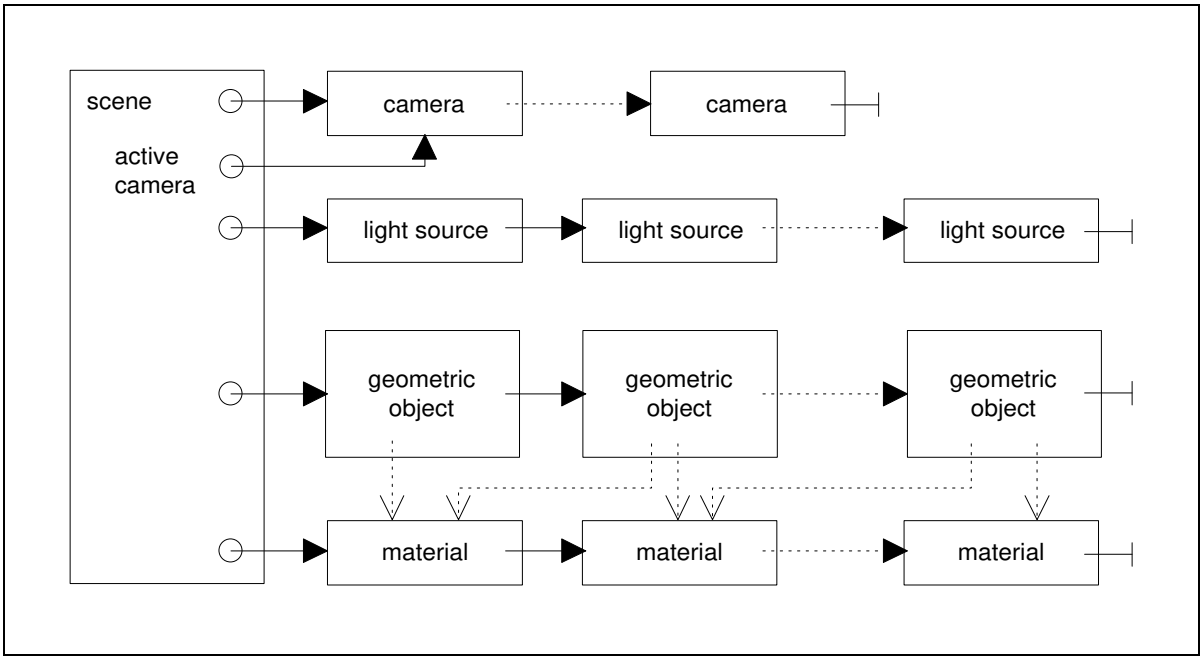

Figure 6.2: Class Scene

Class Scene has to manage the following:

- reading a scene description file (see App. C), and compute the transformation matrices for the object hierarchy
- computation the size of the scene (for movements in relation to the scene size) and an overall bounding box
- keeping a backup of the initial camera for a reset view function
- information about the window aspect ratio to set an appropriate camera
- selection of a drawing mode (wire frame, hidden line, flat or smooth shading)
- selection of an anchor highlighting method, and turning anchor highlighting on/off
- drawing the scene (using the current settings)
- picking objects (for link activation or movement tasks)
- clean up (deleting the lists) whenever a new scene is read, or at program termination

The following excerpt of the class definition shows the public interface of *class Scene*:

```
class Scene
{
  public:
```

```
    Scene ();
    ~Scene ();

    void clear ();                  // destroy scene, free memory

    int readscene (const char* sdffilepath);
    int readscene (const char* path, const char* sdffilename);
    int readscene (FILE*);

    void print (int all = 1);

    // *** drawing ***
    void draw (int mode = -1);          // default: current mode_
    void mode (int);                    // set drawing mode

    Material* findmaterial (const char*);  // find material by name

    // *** size, center ***
    float size () const;                // get the "size" and center
    void get_center (point3D&) const;   // of the bounding box

    // *** picking ***
    GeometricObject* pick_object (point3D&, float tnear,
                    point3D* hitpoint = 0, vector3D* normal = 0);
    GeometricObject* pick_object (float, float, point3D* hitpoint = 0,
                            vector3D* normal = 0);
    void activatelinks (int);           // highlight links or not
    int linksactive () const;           // are links highlighted

    void linkscolor (int);              // set anchor highlighting

    // *** camera ***
    camera* getcamera () const;         // get active camera
    void storecamera () const;          // store active camera
    void restorecamera () const;        // reset camera
    void setwinaspect (float);          // set window aspect
    float getwinaspect () const;        // (width/height)

};// Scene
```

6.5. 3D Objects

All objects of the 3D scene description, namely geometric objects (polyhedra), cameras and light sources, are derived from a common base class, called *class Object*. This class manages the data common to all objects:

- an object number and name, and the number of the *parent* object.
- a transformation matrix from object to world coordinates (see also Figure B.1 in Section B.2.4) and its inverse.
- the channel values for translation, rotation, and scaling used to build the transformation matrix (see App. C.2).

The parent object allows to describe an *object hierarchy*. The position of an object can be given relative to another object. Each object can have exactly one parent or no parent at all, thus the hierarchy is a forest (a collection of trees). All kinds of objects can be included into the hierarchy: for example, cameras can be bound to objects, or light sources can be bound to the camera. When an object has no other purpose than describing transformations within a hierarchy it serves as a dummy object.

This is the public interface of *class Object*:

```
class Object
{
  public:
    Object (int obj_n, int par = 0, const char* name = 0);

    int getobj_num () const;
    const char* name () const;

    enum
    { ch_xtran, ch_ytran, ch_ztran, ch_xrot, ch_yrot, ch_zrot,
      ch_xscale, ch_yscale, ch_zscale, ch_scale,
      num_channels  // number of channels
    }; // transformation channels

    const matrix* gettrfmat ();
    const matrix* getinvtrfmat ();
}; // class Object
```

6.6. Camera Model

For viewing 3D scenes a perspective, untilted camera model is used [Hill90]. A camera is defined by a viewpoint (eye position) and a reference (or lookat) point. These two points together form the line of sight. The distance between viewpoint and viewplane is called focal length. The field of view is determined by the aperture (height of the camera viewport) and the aspect ratio (width/height ratio of the viewport.

Denoting the vector lookat point - viewpoint by n, the other vectors of the camera coordinate system are calculated as follows: $u = n \times up$ (cross product), for an untilted camera the up-vector equals (0, 1, 0), and $v = u \times n$. The vectors u, v, n form an orthogonal coordinate system, u and v lie in the view plane horizontally and vertically, as can be seen in Figure 6.3.

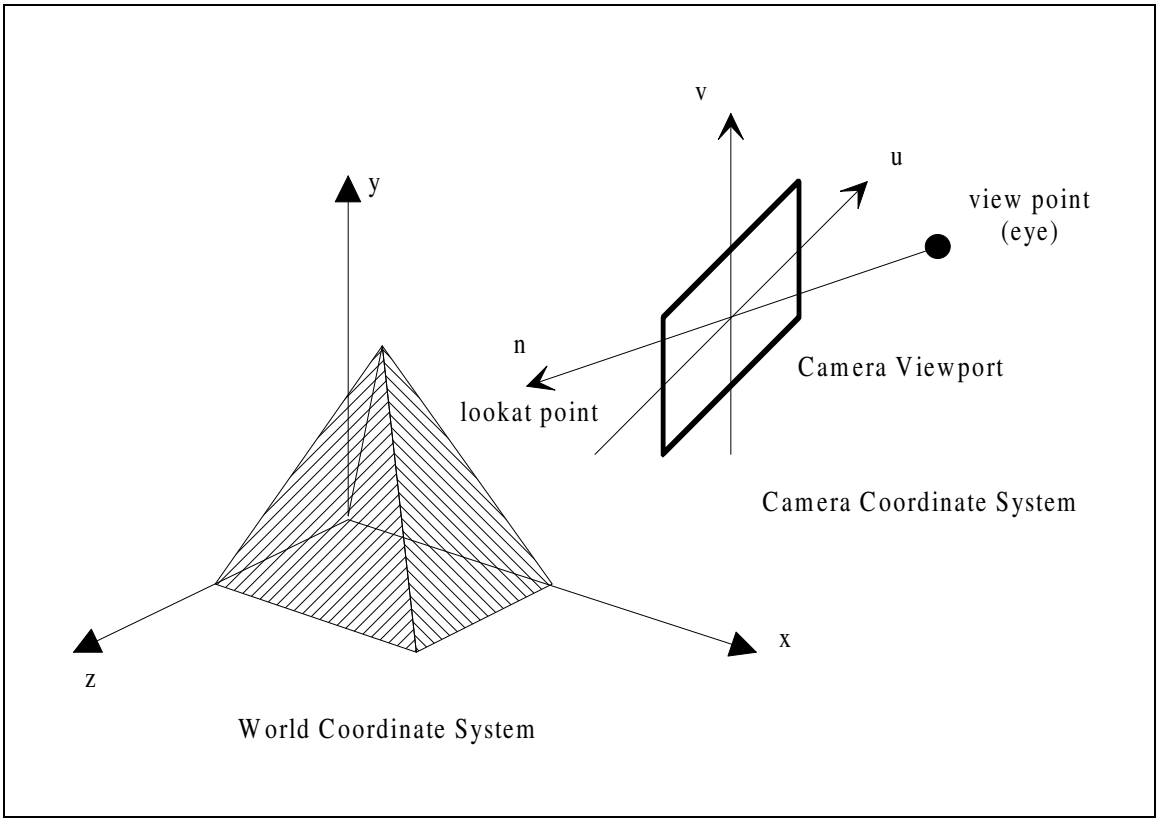

Figure 6.3: The Camera

Depth clipping is achieved via two clipping planes, called hither (near clipping plane), and yon (far clipping plane). For perspective projections the near clipping plane cannot be set to 0 because of the z-Buffer algorithm for hidden surface elimination - it has to be set to as near 0 as possible.

6.7. Light Sources

Light sources are defined by their position and the colour of emitted light. All light sources are treated as positional, diffuse light sources. The amount of light reflected by a face depends only on the position of the light source and not on the current view point. More sophisticated lighting models such as specular light or spot lights may be implemented in future. Currently only very expensive hardware supports such lighting models in real-time.

6.8. Geometric Objects

Currently all objects which comprise a scene are described as polyhedra, or more generally in terms of faces. The same approach is taken by Wavefront's Advanced Visualizer [Wave91].

To allow for possible extensions, an abstract class *GeometricObject* was designed, which describes all possible scene objects. Polyhedra are represented by the subclass Polyhedron. Other kinds of objects, for example spheres, can be derived from the base class. The following methods have to be implemented for geometric objects:

- reading the data from a file (e.g. .obj-file for polyhedra, see Section C.6)
- drawing the object
- picking the object

Common to all geometric objects is the maintenance of bounding boxes (to speed up picking, one in object coordinates and one in world coordinates), as well as a transformation matrix and its inverse for transformations between those coordinate spaces (see App. B.2.4).

The transformation matrix is also necessary for drawing and picking. The draw() routine of class GeometricObject pushes the transformation matrix onto the matrix stack, calls a virtual function draw_() and removes the matrix from the stack. The picking function needs the inverse transformation matrix to transform the ray from world to object coordinates in which the real hit test can be performed (see Section 6.10.).

6.9. Polyhedra

Class *Polyhedron* is derived from class *GeometricObject*. A polyhedron consists of:

- an array of vertices
- an array of vertex normal vectors (for smooth shading)
- and an array of faces.

The type *face* is defined in the GE3D graphics library (see App. B.1). Each face contains of an array of integer indices into the vertex array and (optionally) into the normal vector array.

The definition of face *materials* (colour) is not part of the face data structure, for reasons of space and time efficiency (many faces have the same colour) and for possible future extensions. To allow different materials for the faces, class Polyhedron also maintains a list of groups of faces with the same material. Each such material group has a pointer to the first and last face it comprises and a pointer to the actual material definition.

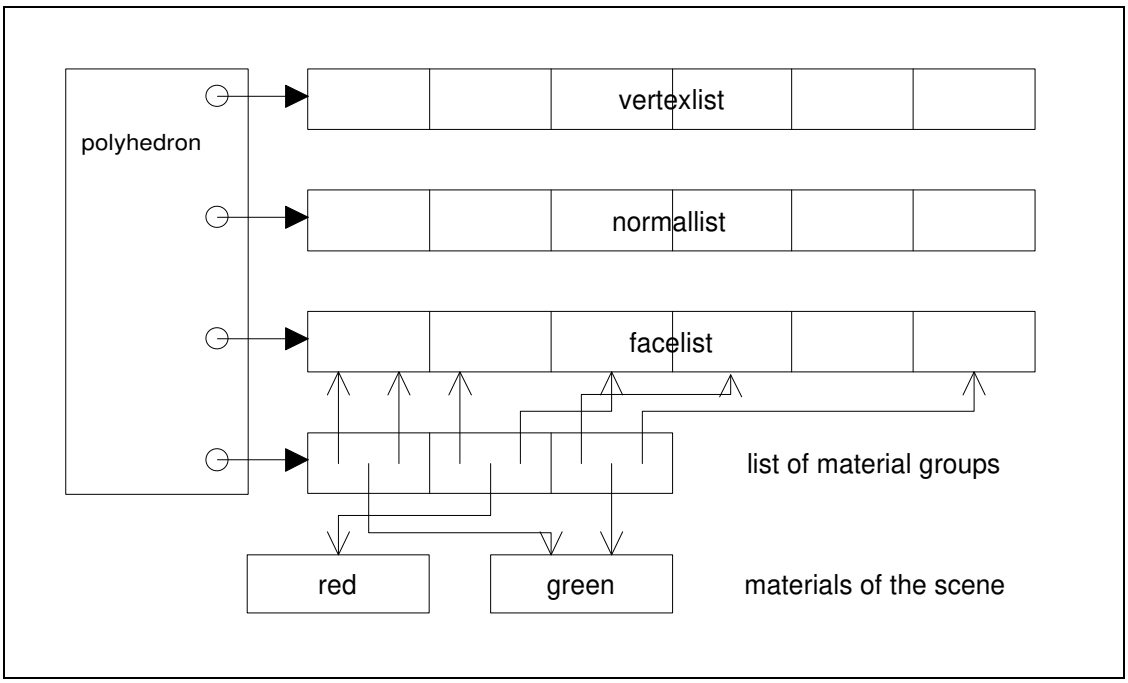

Figure 6.4: Class Polyhedron

In Figure 6.4 there is one face of material green, two red ones, and again three green faces. For drawing the material group list is traced, whereas the hit test function for picking (see next chapter) accesses the facelist array since it does not need to take face properties into account.

For space efficiency data is shared by objects of the same type. In 3D scenes, often one and the same object is placed several times in the scene, for example chairs and tables in a room. These polyhedra have to be stored as separate objects, because they have their own transformations that describe their positions in the scene. All other data including vertex and normal list, the face list and list of material groups can be shared between the objects. The data has to be read and stored only once.

6.10. Picking 3D Objects

Picking means to determine which object in the scene was hit when the user clicks on a point on the window. These algorithms are needed in two situations in the 3D viewer: first for the activation of links. When an anchor object is clicked it must be determined which one was hit in order to activate the link. Second, picking is necessary for point of interest movement (see Section 6.13.). There the exact coordinates and the outward normal vector in the hit point are needed.

Figure 6.5 illustrates the coordinate systems in which the ray from the view point to the hit point can be seen:

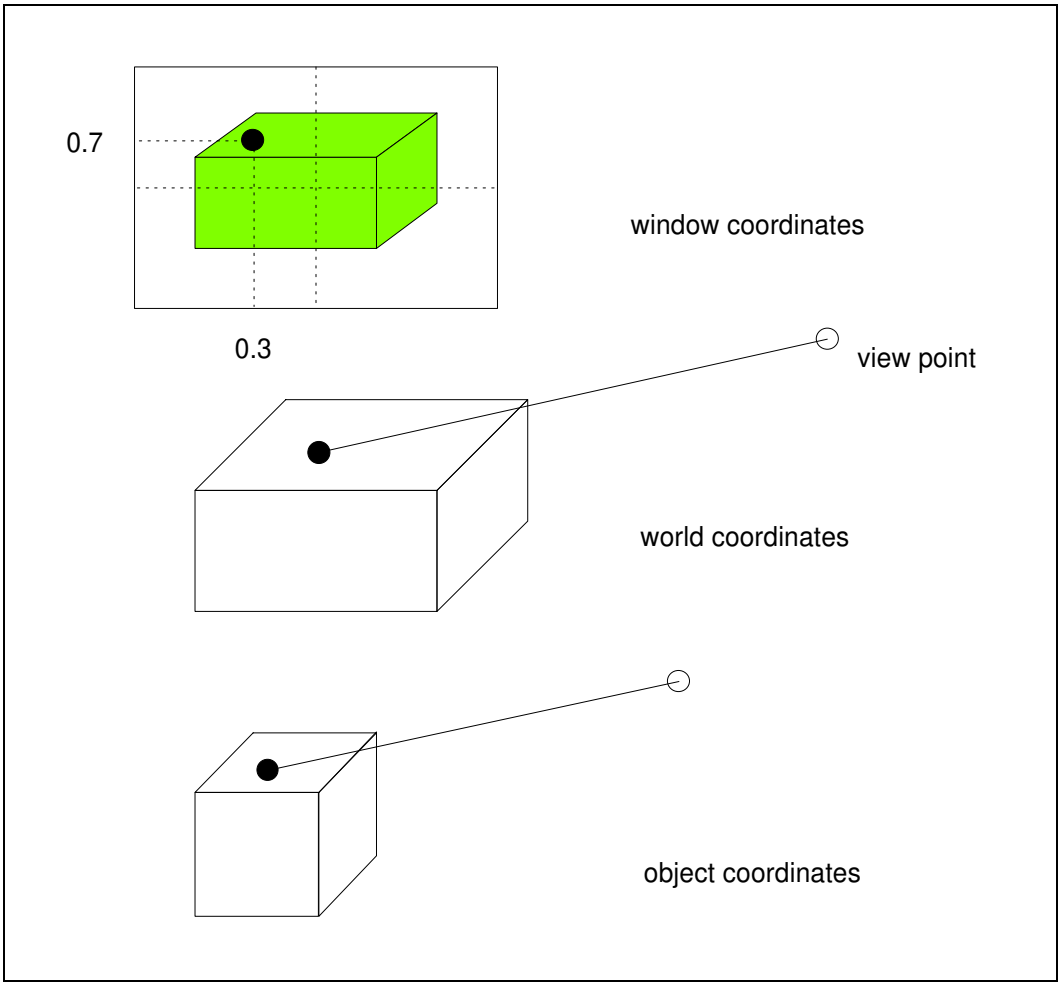

Figure 6.5: Ray for Hit Testing in Different Coordinate Systems

Note that the ray projects to a single point depending on the current perspective projection. The input to the hit test is the position of the point in normalised window coordinates (e.g. 0.3 width, 0.7 height in the example in Figure 6.5) and the current camera settings. Several steps are necessary to determine the object hit, the hit point in world coordinates, and the normal vector of the face hit:

1. Calculation of the ray

Rays are described by the formula A + t*b, where t is >= 0, A is the start point and b the direction of the ray. Here A is the view point, b the vector from view point to the lookat point. The object which the ray hits for the minimum value of t has to be found. To consider only *visible* faces, t must be greater than or equal the distance of the near clipping plane.

2. Test with bounding cube

Before the ray is tested with each object it is tested with the bounding cube. A ray can intersect an object only if it intersects its bounding cube. As a special case the object can

also be hit when the ray starts inside the bounding cube, e.g. when the camera is inside a room. These fast initial tests considerably reduce the number of necessary hit tests.

3. For each object: transformation to object coordinates

To test the ray with each geometric object (in our case a polyhedron) it is transformed to object coordinates by applying the inverse transformation matrix. Lets call the new ray A´ + t*b´. Note that the ray has the *same* parameter *t*.

4. For each face: the hit test

The next step is to test whether the ray hits the *plane* in which the face lies from outside at a value of t less than the current minimal value of t. If so, the face is a candidate for the hit. It has now to be tested whether the hit point lies inside the face.

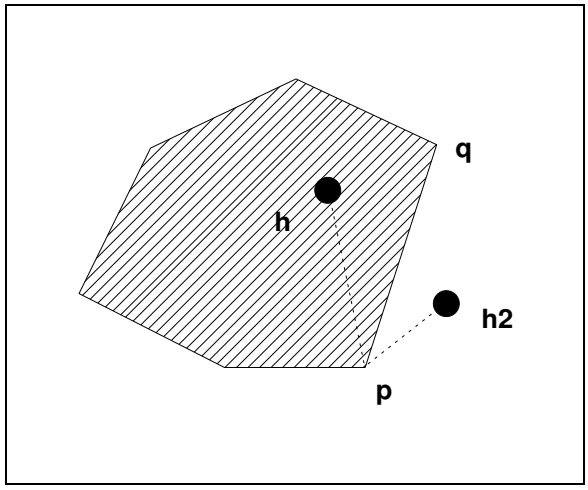

Figure 6.6: Hit Test for a Convex Polygon (outward normal n out of page)

For *convex* faces this is simple, since convex polygons are the intersection of half planes. The hit point has to lie on the correct side of each polygon edge. With the names of Figure 6.6 a test of the hit point h against the edge p-q means to test whether $\langle (q - p) \times (h - p) \cdot n \rangle$ is greater or equal zero. Recall that for the cross product c = a x b, the vectors a, b, c form a right handed coordinate system, and that the scalar product $\langle n \cdot m \rangle$ is greater than zero if and only if the angle between n and m is less than 90 degrees. In this example n and (q - p) x (h - p) point out of the page, whereas (p - p) x (h2 - p) points into the page.

5. Calculation of Hit Point and Normal Vector

The hit point is given by A + thit*b (in world coordinates). The normal vector has to be transformed from object to world coordinates. A little vector arithmetic shows that this is done by applying the transposed inverse transformation matrix.

6.11. Materials

A *class Material* has been designed to cover all properties of faces that can be rendered. Currently only the RGB component of the diffuse colour is stored in Class Material, but the concept is extensible, e.g. for specular colour and textures (see lighting models in Chapter 6.7.), as the possibilities for real-time rendering increase.

As mentioned in Chapter 6.4., class Scene keeps a list of materials. Each material has a name (the one read from file) which is the name under which the material was edited with Wavefront's Property Editor. These names are referenced for groups of faces of polyhedra.

Class Material is also responsible for setting the appropriate colour according to the current anchor highlighting mode. For example in mode *brightness* a (pre-calculated) brighter or darker shade of the natural colour (depending whether the object is pickable or not) is used for a call to ge3d_setfillcolor() of the GE3D graphics library

6.12. Movement

All navigation modes are implemented in terms of camera movements. For *flip object* movements of the object have to be simulated by movements of the camera. For translations this is only a change of the sign, and rotations have to be done around the centre of the scene instead of the viewpoint. The user gets exactly the same picture as if the object were moved, and no special treatment of this mode is necessary. There is one disadvantage: as an untilted camera is used, it is not possible to turn the object around any axis (rotations around the z-axis are impossible).

Walk/heads-up: the motion function for translation and rotation was chosen as $\lambda x + \mu x^3$ for appropriate values of λ and μ, where x denotes the distance the mouse was dragged as fraction of the window size. For small movements λx dominates, resulting in a fine movement. For larger distances μx^3 dominates, allowing fast movements over larger distances. x^3 is used, because it is easy to calculate and preserves the sign.

Heads-up: Individual translations and rotations (horizontal/vertical) are handled the same way as in *walk*. Only the activation and mouse button assignment differs.

6.13. Point of Interest Movement

Point of interest movement [Robert93, Mack90] is the movement technique for *fly to* (see Section 5.3.4.). The idea is to make the movement towards the point of interest (POI) proportional to the current distance from it. If the distance at time t is denoted with d (t) and the movement with m (t) we get the relation m (t) = k*d (t) for a suitable constant k. Figure

6.7 shows the resulting movement of one time step along a straight line towards the point of interest.

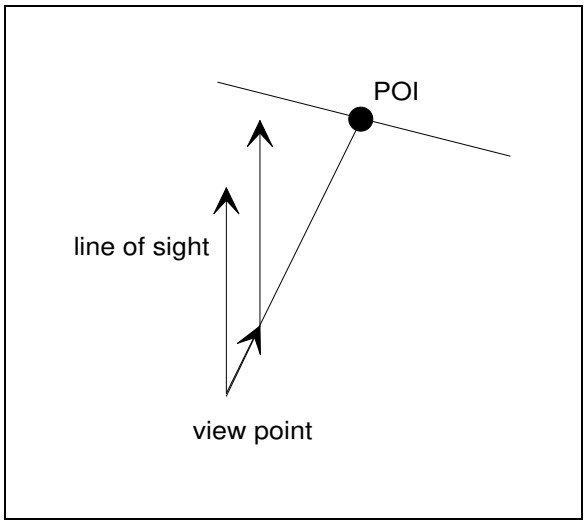

Figure 6.7: Point of Interest Movement

When taking into account that the change of the distance is the negative movement (positive movement decreases the distance) we obtain d' (t) = - k*d (t) for the derivation d'. The solution of this simple differential equation is $d(t) = d_0 * e^{-kt}$, a logarithmic motion function as can be seen in Figure 6.8. d_0 is the distance at t = 0.

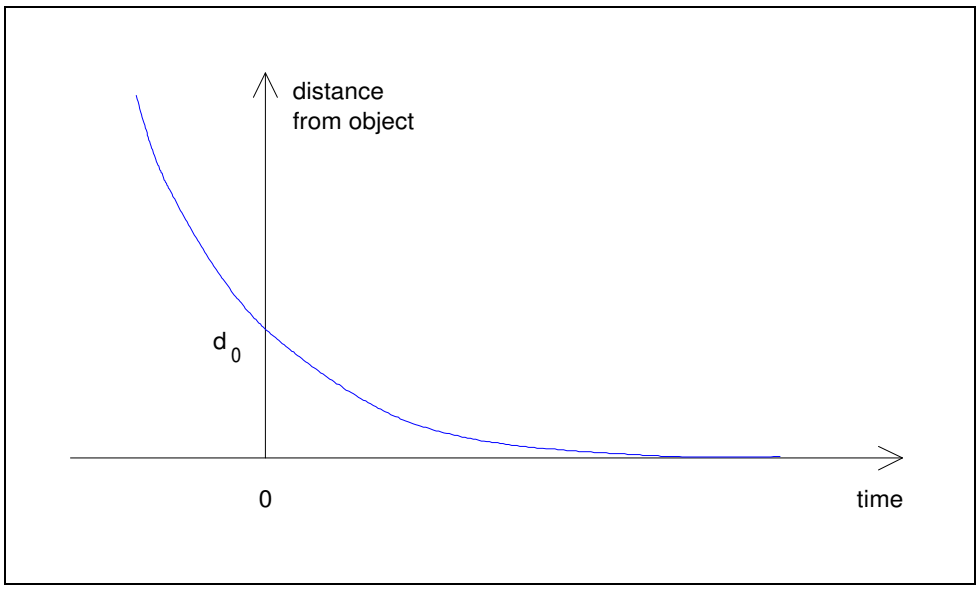

Figure 6.8: resulting distance on motion towards POI

Figure 6.8 shows that the approach to the point of interest is fast at the beginning, but it always remains controlled, and a collision never happens. In the 3D viewer the motion is stopped at the near clipping plane (otherwise the point of interest would get invisible). Also remarkable is that the point of interest always projects to the same point on the window.

As an extension to the basic POI-movement, a *rotation* of the line of sight to face the normal vector is also possible, as shown in Figure 6.9. This is useful to approach objects head on.

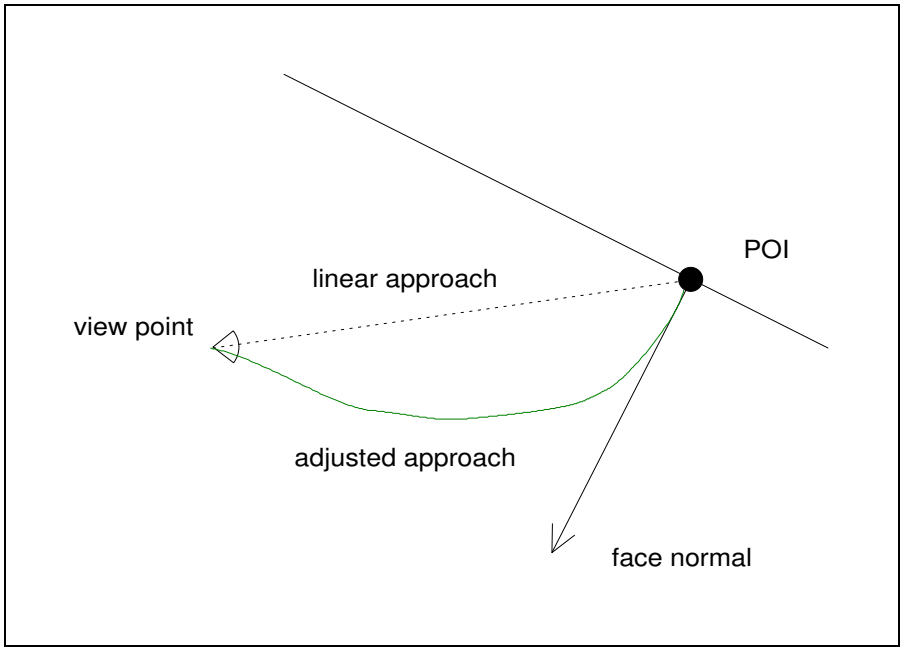

Figure 6.9: Point of Interest-Movement With Rotation

The movement is composed of two parts: a lateral movement towards the face normal (this moves the point of interest towards the window centre) and a linear zooming towards the point of interest.

6.14. Anchor Highlighting

- **mode "brightness"**

To modify the brightness of objects, first the colour of each face is transformed into the HLS colour space [Foley90, Hill90]. Then the L parameter (lightness) is transformed linearly into the interval [0.7, 1] for anchors and [0, 0.3] for other objects. The resulting HLS triple is then transformed back to RGB values.

When adding black or white in the RGB model, colours become greyish. The transformation via the HLS colour model is better as it preserves the saturation (S).

- **mode "colour code"**

The brightness for monochrome non-anchors is calculated as $Y = 0.299\,R + 0.587\,G + 0.114\,B$. This is the brightness parameter Y of the YIQ model [Foley90], which is also used

for monochrome TV sets. It takes into account the different sensitivity of the human eye for red, green, and blue light and is therefore a scale of the *seen* brightness.

6.15. Colour Chooser

Colours can be modified with a *colour chooser dialogue*, see Figure 6.10 (or colour plate 9). It has scrollbars for both RGB and HLS colour model [Foley90], which are updated simultaneously as well as a rectangle showing the current colour.

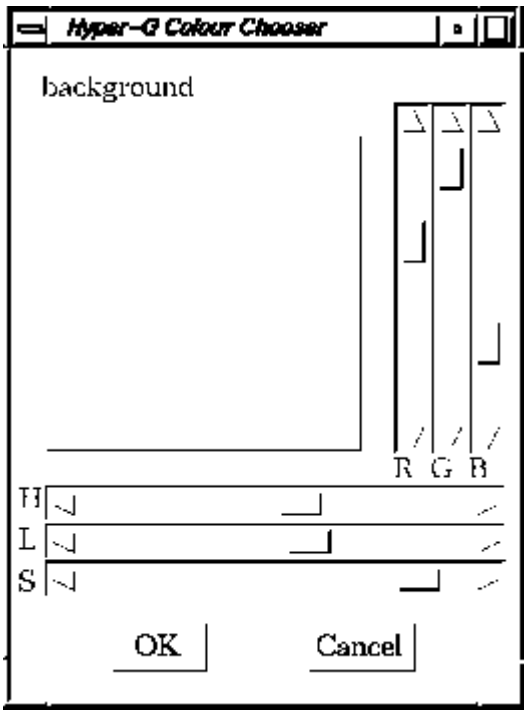

Figure 6.10: Colour Chooser Dialogue

7. Usability Evaluation

To evaluate the implementation of the Hyper-G 3D viewer, sixteen students were asked to do a *heuristic evaluation* [Nielsen90] of the program. Many of the suggestions they made were implemented.

After the heuristic evaluation, two *usability tests* [Nielsen93] were performed for each of two important features of the user interface:

- navigation methods and
- anchor highlighting.

The tests produced comparable results for the diverse methods implemented. They also gave hints for improvements or program changes. Questionnaires filled out by the test persons reflected a wide range of comments and opinions.

Four independent groups - two for navigation and two for anchors - each tested 8 *novice* and 8 *expert* users as part of the Human-Computer Interaction practical classes in summer semester 1993. The line drawn between novice and expert users was the familiarity with graphical user interfaces (GUIs). All expert users were familiar with and frequent users of at least one GUI like MS-windows, Apple's UI or X-windows. Novice users had no such experience.

The usability tests were designed so that they could be carried out in about half an hour per test subject. Of course, novice users needed more time than expert users. During the test one test supervisor explained the program and gave help as necessary, and a second recorded objective data and any remarks made by the user.

The results of the heuristic evaluation and the usability tests are discussed separately for navigation (Chapter 7.3.) and anchors (Chapter 7.4.).

7.1. Test Environment

The heuristic evaluation and usability tests were done with a stand-alone version of the program, not integrated into Hyper-G but started separately. The main reason for this is that users should concentrate on testing this viewer and not the whole Hyper-G system. A second reason is that the other viewers under the X-windows environment are currently under implementation and subject to changes.

Where viewers for text documents or raster images were needed, a simple viewing window for that document type was opened by the 3D viewer itself.

The tests were performed on Silicon Graphics workstations. Two groups worked on a Personal IRIS 4D 35/TG and two on an IRIS INDIGO XZ4000.

Several scenes were modelled for the test environment: models of cars, rooms, and offices, a model of an imaginary building with several rooms, which was used for most of the tests - a wireframe model can be found in Colour Plate 10 in App. A) -, and a model of the IICM institute (see Colour Plate 4).

7.2. Heuristic Evaluation

The heuristic evaluation was done by four groups of four students. Two groups concentrated on the navigation methods, the other two on anchor highlighting methods. After an introduction into the system each student carefully examined the program to find potential usability problems and possible improvements.

The individual results were then combined into a group aggregate. A discussion of these results can be found in Section 7.3.1. for the navigation modes and 7.4.1. for anchor highlighting.

The evaluators complained of the lack of a built-in help system in the prototype 3D viewer. Help will in fact be integrated into Hyper-G: on-line help with cross-references is just a special case of hypertext.

7.3. Navigation Modes

As discussed in Chapter 5.4. the 3D viewer offers one mode for object manipulation, called *flip*, and four alternatives for navigation: *walk*, *fly*, *fly to*, and *heads-up*. This section discusses the results of the heuristic evaluation and the usability tests concerning navigation modes. The heart of the evaluation is the comparison of the four navigation modes, which can be found in Section 7.3.9.

7.3.1. Heuristic Evaluation of Navigation Modes

Some general facts and suggestions apply to navigation in general, others are specific to one of the navigation modes *flip object*, *walk*, *fly*, *fly to*, and *heads-up*.

- **navigation (general)**

- ”Walking or flying through walls is irritating.“
  A solution to that problem would further slow down movement since it would involve continual hit detection in software. Note that motion mode "fly to" does not have this problem.

- ”A *status line* displaying the current navigation mode and a short description or explanatory symbols would be useful.“

- ”A *navigation window* could display the current position of the camera. Front, top and side views can show the viewpoint and line of sight.“

- ”An undo function for movements would give more safety and encourage experiments.“
  All motion modes are in fact easily reversible.

- ”Different mouse cursors should reflect the current program state and the expected input actions.“
  The mouse cursor changes its shape in important cases, for example when *fly* or *fly to* is active or when the user has to wait for another document (hourglass). Changing the mouse cursor on any mouse dragging, e.g. in *walk*, would be possible.

- **mode "flip object"**

- ”Stationary scene elements like coordinate axis or a room could emphasis that the object is being moved and the position of the viewer does not change.“

- ”The object behaves strangely when viewed from straight above or below.“
  This problem is due to the untilted camera model (see Section 6.6.).

- ”The direction of mouse movement for zooming should correspond to the object movement in the same way as for translation and rotation.“
  This bug was immediately fixed. Previously the mouse could be dragged from the bottom or left to the top or right for zoom in. Now dragging the mouse up moves the object away (zoom out) and vice versa.

- **mode "walk"**

- ”The mouse sensitivity should be the same for all movements, and the sensitivity should be changeable by the user.“
  It was improved by trying various values. The sensitivity can be regulated via the X-attributes; a dialogue was not implemented, since these values are rarely changed by a user.

- **mode "fly"**

- ”A slider should be used for regulating the flight speed.“
  This would it make impossible to control speed and direction of flight simultaneously. To manipulate the slider, movement would have to be turned off - otherwise the fly direction would get out of control. An alternative would be to use keyboard commands for acceleration.

- ”You should only fly as long as a mouse button is pressed (held) down; this would increase the feeling for the user of controlling the system.“
  That is true, but *fly* was designed to allow continuous motion without holding any mouse button - and is appreciated by some users for only that reason.

- **mode "fly to"**

- ”*Fly to* as a mode of its own is not effective (turning is only possible when an appropriate face is visible) but is very useful when embedded in *heads-up*.“

- ”*Fly to* is also a useful additional feature of *flip object*.“

- ”Automatic view orientation during *fly to* is sometimes disturbing. Often a straight movement towards the point of interest is more appropriate. This applies especially to flying backwards.“

- **mode "heads-up"**

- ”When activating *fly to* the marking (clicking) of a point of interest should not be irreversible.“
  The implementation was changed such that the POI can be repositioned as often as desired.

- ”The line drawn from the active icon to the mouse cursor is somewhat irritating.“
  Consider that the line also shows the amount of movement and allows to undo the movement by dragging the mouse back to the start point.

- ”The walk-icon must symbolise motion.“
  It has been changed to a walking person, rather than a stationary person.

7.3.2. Usability Tests of Navigation Modes

As an introduction to the program - mainly for novice users - test persons were presented a model of a car (the VW beetle, see Colour Plate 2). With navigation mode *flip object* users should familiarise themselves with using the mouse for 3D manipulations. Two simple exercises with generous assistance were given for that purpose: rotating the car around its vertical axis and zooming to the front light.

The main purpose of the usability test was to obtain *comparable* results for the four navigation modes that move the viewpoint in the scene. Each of *walk*, *fly*, *fly to*, and *heads-up* was explained to the user in a demo scene (a room). When subjects felt able to apply the mode themselves in another scene the real test began.

For each mode the test subject had to walk into an imaginary building, turn once left and once right and looking under the table in that room. There was a name written on the bottom side of the table the user had to read. This task ensures that the users had to have good control over each mode and that the label could not be read by accident. The time needed to perform each exercise was used as an *objective* criterion for comparison.

To round off subjects could navigate through a model of the IICM institute with a free choice of the motion mode. As a "reward" there are some pictures of institute members in the foyer. Test subjects were free to explore the institute further, if they liked.

To measure *subjective* user satisfaction with the modes, a questionnaire was used. The questions below were asked for each movement mode and the user made a cross on the bar resulting in marks on a scale from 0 to 10. Other questions concerned the satisfaction of using the program as a whole, remarks to each mode and suggestions for improvements. Finally users were asked to rank the four motion modes or at least say which one they liked best.

How easy is *xxx* to learn? very hard very easy
O---o---o---o---o---O---o---o---o---o---o---O

How much did you feel to have control over movement with *xxx*?	very poor very much O---o---o---o---o---O---o---o---o---o---o---O
How useful (over all) is mode *xxx*?	very useful unusable O---o---o---o---o---O---o---o---o---o---o---O

The discussion of each navigation mode (except *flip object*) contains a table of the time needed by novice and expert users and the results of the questionnaires (the question of movement control was only made by one of the two groups).

7.3.3. "Flip Object"

As this mode is a technique for *object* movement, it would have made no sense to compare it with the navigation modes, where *navigation* is more strictly taken to mean positioning of the viewpoint. During the introductory exercise the following observations were made:

- Most test persons preferred to use the buttons "translate", "rotate", "zoom" instead of the three mouse buttons.
  This allows performing all movements with just the left mouse button.

- ”*Fly to* is a useful addition to *flip object*.“

The scene used for this test was a model of a VW beetle (see Colour Plate 2).

7.3.4. "Walk"

The following statements were made by the test persons:

- ”It is difficult to remember the functionality of the mouse buttons.“
  A status window displaying icons would be helpful. Alternatively the cursor could change when pressing a mouse button.

- ”The motions are jerky and slow when using walk.“
  In the other modes (which are in fact not faster) users did not feel that. One possible reason is that the other modes all display some information on the window for users to concentrate on.

- ”Walking through walls is very disorienting.“
  The same problem also arises in *fly*. Note that *fly to* provides a solution to this problem.

- The tasks were performed easily and fast using *walk*, but its subjective evaluation was rather bad.
  Probably this is due to the memory load for remembering the correct mouse buttons, but with some practice *walk* is efficient to use.

Table 7.1 shows the quantitative results obtained in the usability tests for navigation mode *walk*.

walk	time	ease of learn	control	usability
novice users				
average	4.13	6.47	7.50	6.43
std. deviation	2.84	2.75	2.35	2.15
expert users				
average	3.21	6.94	7.75	6.45
std. deviation	3.11	2.68	3.23	2.28
all users (average)				
average	3.67	6.71	7.63	6.44
std. deviation	2.98	2.72	2.84	2.22

Table 7.1: Evaluation of *walk* [time in minutes, answers 0 to 10 points]

Against expectations, novice users did not need much more time than expert users to perform the task. The control over movement is said to be good.

Although it was thought that *walk* would not appeal to novice users, it turned out that they managed this mode well. They rarely used a mouse button other than the left one (since they forgot the purpose of the other ones), but they were able to wander around well and were quite satisfied.

One of the novice users did not succeed at all with *walk*. He had serious problems and the test had to be continued with the next exercise.

7.3.5. "Fly"

The following observations were made:

- Many novice but also some expert users had problems controlling speed and direction of flight. On the other hand, a few users were very enthusiastic about this mode - they remarked it was even easier to use than *walk* or *heads-up*.

- Some users had difficulties in frequent clicks with all of the three mouse buttons. When they wanted to stop the flight they could not "find" the left mouse button in time and so they lost orientation.
  As a solution the flight could be made active only as long a mouse button is pressed (against the concept of fly). Or cursor keys could be used to control speed (involving the second hand).

- ”Flying backwards is quite unnatural.“
  But it is an easy way to reverse overflights.

The quantitative results obtained in the usability tests are summarised in Table 7.2.

fly	time	ease of learn	control	usability
novice users				
average	6.59	5.50	7.13	5.19
std. deviation	3.45	2.98	1.45	3.08
expert users				
average	3.66	5.26	6.13	5.98
std. deviation	1.96	3.50	2.85	2.61
all users (average)				
average	5.13	5.38	6.63	5.59
std. deviation	2.71	3.24	2.15	2.85

Table 7.2: Evaluation of *fly* [time in minutes, answers 0 to 10 points]

Although some had problems, all test persons completed the exercise.

Obviously this mode appealed more to expert users. Novice users needed almost twice as long as expert users for the exercise. Nevertheless expert users gave this mode a worse mark for movement control than novice users, and the high standard deviation indicates the wide range of opinions. Most users agreed that this mode is rather difficult to learn and proves not so useful as other modes.

7.3.6. "Fly to"

The following problems of *fly to* were detected:

- The main problem of *fly to* is performing rotations. Also, a means of vertical translation to change the height of the viewpoint is considered lacking.

- ”*Fly to* is very well suited as an addition for *flip object* and *heads-up*, but is not sufficient as a navigation mode in itself.“

- Automatic viewpoint orientation towards the normal vector is sometimes disturbing - although quite liked to obtain detail views. Using the SHIFT-key to suppress it was not accepted by the test subjects.
  This adjustment has been turned off by default afterwards, in particular backward movement is more natural without rotation.

- ”Holding a mouse button to perform a movement is rather unconventional.“
  An alternative use of cursor keys would require to use both hands or (even worse) to change between keyboard and mouse.

Table 7.3 shows the quantitative results obtained in the usability tests for navigation mode *fly to*.

fly to	time	ease of learn	control	usability
novice users				
average	9.96	7.40	6.63	7.59
std. deviation	5.32	2.86	3.39	3.19
expert users				
average	4.89	7.74	6.00	6.79
std. deviation	4.17	2.62	3.32	3.01
all users (average)				
average	7.43	7.57	6.32	7.19
std. deviation	4.75	2.74	3.36	3.10

Table 7.3: Evaluation of *fly to* [time in minutes, answers 0 to 10 points].

Two out of 16 novice users did not manage the task with *fly to*. They had serious problems performing rotations or repositioning badly placed points and flew too far towards the marked point. The other novice users needed a very long time to do this exercise, and often a reset to the initial position was necessary.

Due to a bug in the program present at time of the usability tests, the point of interest could not be repositioned under certain circumstances (see also Section 6.13. point of interest movement). This probably lead to worse marks for this navigation mode. These problems aside, test subjects agreed that *fly to* is easy to learn.

Fly to got rather bad grades for movement control but was used very often in *heads-up*. The high standard deviation indicates that some users used this mode quite well, some even flew through two or three rooms at once. Also the task of reading the label under the table turned out to be very easy with this mode: some few mouse clicks and it was zoomed to fill the window.

7.3.7. "Heads-up"

These were the opinions of the test persons:

- The main advantage of *heads-up* is that only very few things have to be remembered. All motion can be done with the left mouse button (apart from zooms with integrated *fly to*).

- ”The icons give a feeling of security.“
  It is intuitively clear how to use the mouse to obtain the desired motion.

- Many users were not flexible enough to choose the right icon appropriate to the desired movement.

From observations during the tests the majority of test persons fall into one of the following groups: About one third (almost) exclusively used the "walk" icon, the other third selected *fly to* (and nothing else), whereas the last third made alternate use of *fly to* and the "eyes" icon (for rotations).

The quantitative results obtained for *heads-up* can be found in Table 7.4.

heads-up	time	ease of learn	control	usability
novice users				
average	4.64	7.20	8.50	7.71
std. deviation	1.82	2.63	1.12	1.91
expert users				
average	2.62	7.47	7.00	7.19
std. deviation	1.11	3.07	2.92	3.40
all users (average)				
average	3.63	7.34	7.75	7.45
std. deviation	1.47	2.85	2.02	2.66

Table 7.4: Evaluation of *heads-up* [time in minutes, answers 0 to 10 points].

All users were able to perform the test without difficulties. They also agreed that this mode is very easy to learn and gives high control over movement; especially novice users felt very safe. Overall the usability of *heads-up* is very high.

Expert users were better able to use all the possibilities of *heads-up* and needed only a very short time for this exercise - many did the task in less than one minute.

7.3.8. Free Choice of Mode

In this test, the subjects had free choice of navigation mode. The test persons could enter and walk around a model of the IICM institute as they liked.

- To get into the building, usage of *fly* and *fly to* dominated. Some test persons used *fly to* with automatic viewpoint orientation. They were surprised to fly through all floors when marking a point on the floor of the foyer.

- For navigation around one floor *walk* and *heads-up* were used most often.

- Most of the expert users (and few novice users) tried to walk upstairs, which is a rather complicated task. *Fly to* (sometimes in combination with the "eye" icon of *heads-up*) and *fly* were used mainly and only few users had problems.
  One test person found the easiest way: he selected the translation icon of *heads-up* to move the viewpoint up to the first floor and walked around as if nothing happened.

7.3.9. Comparison of Modes

As *flip object* is the only mode which moves the object, there is no sense in comparing it with the other motion modes. This section compares the four navigation modes that move the view point: *walk*, *fly*, *fly to*, and *heads-up*.

The first Table summarises the average time needed by the test persons to do the desired task in minutes. As a reminder the task was to enter a room, turn into another one, look under the table and read a name written on it (see Section 7.3.2.). The times are given in minutes (decimal).

time	walk	fly	fly to	heads-up
novice	4.13	6.59	9.96	4.64
expert	3.21	3.66	4.89	2.62
average	3.67	5.13	7.43	3.63

Table 7.5: Average Time Needed for the Various Motion Modes [minutes].

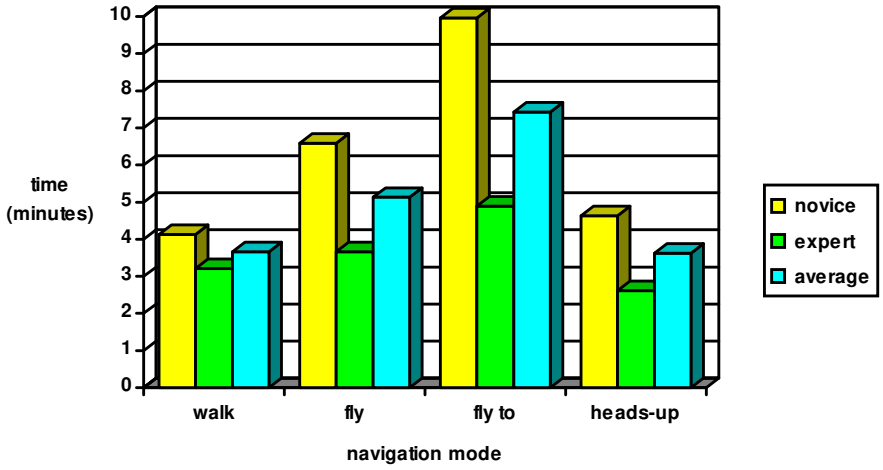

Figure 7.1: Time Needed per Exercise

As can be seen from Table 7.5 and Figure 7.1 especially novice users in particular (but also experts) had most troubles with *fly to*. This was because they had difficulties in finding an appropriate face to perform rotations. Nevertheless *fly to* was used very often when integrated into *heads-up*.

In all modes except *walk* expert users were significantly faster than novice users - they succeeded in about half the time. In *walk* the difference is far less significant - both novice and expert users used *walk* effective. It is amazing that novice users were faster than expert users using *walk*.

The following three tables summarise the data about each motion mode obtained from the questionnaires. The results have been normalised to a scale of 0 to 10, 10 being the highest mark.

The first question was how easy it is to learn each navigation mode. The second was about the control over the motion. And the third was to tell how good the mode is over all.

ease to learn	walk	fly	fly to	heads-up
novice	6.47	5.50	7.40	7.20
expert	6.94	5.26	7.74	7.47
average	6.71	5.38	7.57	7.34

Table 7.6: Ease to Learn the Motion Mode [0 to 10 points].

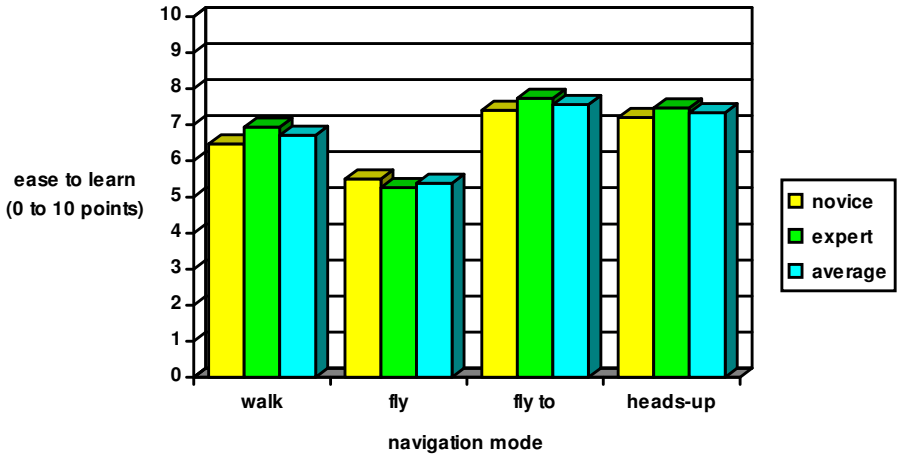

Figure 7.2: Ease to Learn

Table 7.6 and Figure 7.2 show that novice as well as expert users found that *fly to* is the easiest mode to learn, closely followed by *heads-up* and *walk*. The test subjects also agreed that *fly* is the most difficult to learn.

movement control	walk	fly	fly to	heads-up
novice	7.50	7.13	6.63	8.50
expert	7.75	6.13	6.60	7.00
average	7.63	6.63	6.32	7.75

Table 7.7: Control Over Movement in Each Mode [0 to 10 points].

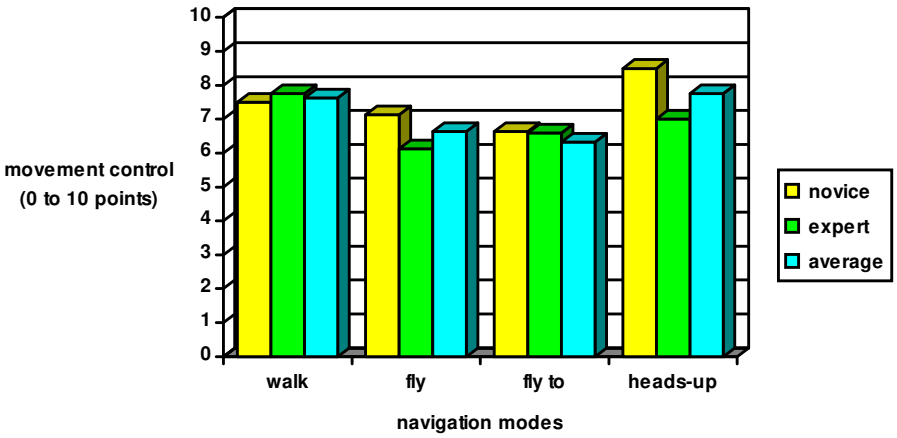

Figure 7.3: Movement Control

Although, in general, test persons reported that *walk* and *heads-up* give a greater feeling of control over movement than *fly* and *fly to*, there are differences between novice and expert users that can be seen from Table 7.7 and Figure 7.3: Novice users favour *heads-up* while experts favourite *walk*. Again it can be seen that novice users had problems with plain *fly to*.

usability	walk	fly	fly to	heads-up
novice	6.43	5.19	7.59	7.71
expert	6.45	5.98	6.79	7.19
average	6.44	5.59	7.19	7.45

Table 7.8: Overall Usability per Navigation Mode [0 to 10 points].

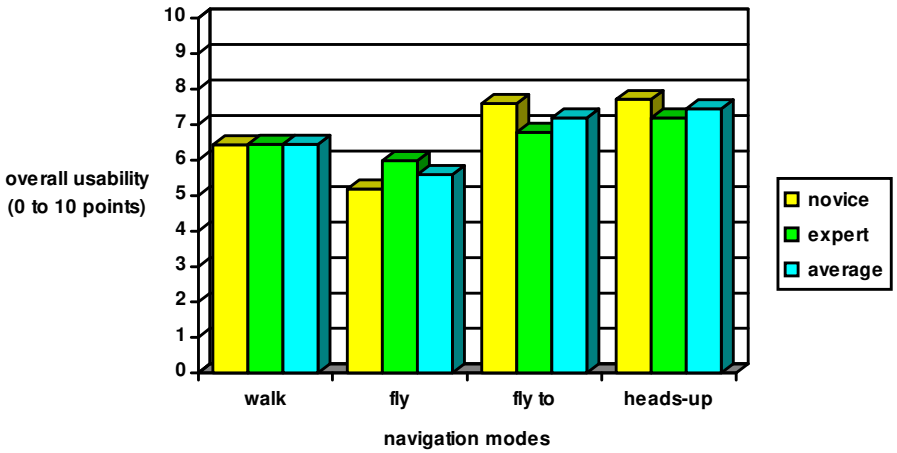

Figure 7.4: Overall Usability (up to 10 points)

Table 7.8 and Figure 7.4 show that novice and expert users gave the same ranking for usability: they regard *heads-up* to be the best mode, followed by *fly to* and *walk*. They also

agreed that *fly* is rather hard to use, although some few (expert) users enjoyed *fly* more than any other mode.

Finally, test persons were asked for their favourite navigation mode - the mode they liked best and which they will use when able to choose freely.

favourite mode	walk	fly	fly to	heads-up
novice	13.3	13.3	33.3	40.1
expert	11.8	23.5	23.5	41.2
average	12.6	18.4	28.4	40.6

Table 7.9: Favourite Navigation Mode [in percent].

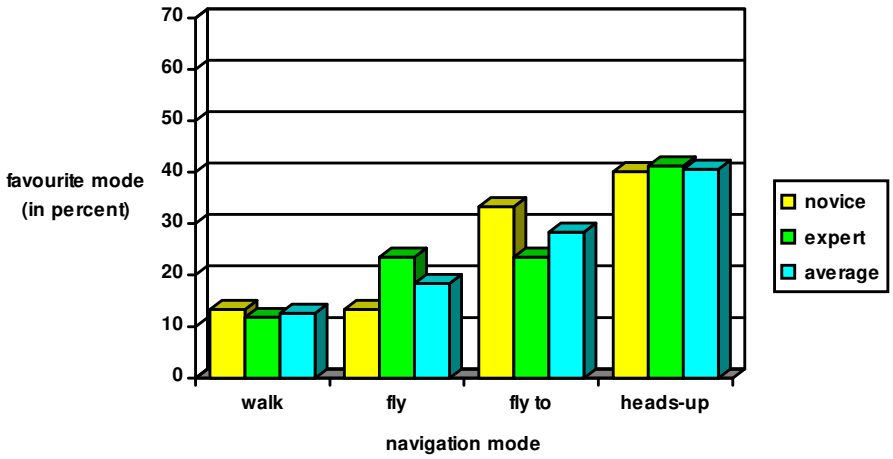

Figure 7.5: Favourite Navigation Mode

The clear favourite is *heads-up*: Table 7.9 and Figure 7.5 show that more than 40 % of all test persons prefer this mode. Strangely, novice users in particular prefer *fly to*, in spite of the problems they had using this mode. Interesting is the relative preference for *fly*, compared to its rather bad marks: almost one in four expert users and nearly 15 % of novice users say that they prefer *fly*. On the other hand *walk* is not as liked, although users achieved good objective results with it.

In summary, Table 7.10 shows the ranking of the navigation modes obtained by the tables above. For a better overview it does not distinguish between novice and expert users. The last line was obtained by combining the ranking of time and the results of the questions.

ranking	walk	fly	fly to	heads-up
time	2	3	4	1
ease to learn	3	4	1	2
movem. control	2	3	4	1
usability	3	4	2	1
favourite	4	3	2	1
overall	3	4	2	1

Table 7.10: Ranking of Motion Modes [1 = best].

7.4. Anchor Highlighting

To isolate anchor testing from the navigation modes, all tests concerning anchors were done in navigation mode *heads-up*. Test persons were able to concentrate on the highlighting of anchors and link activation without distraction.

7.4.1. Heuristic Evaluation of Anchor Highlighting Modes

The 16 students performing the heuristic evaluation examined the different highlighting modes for anchors, reported which ones they think easy and clear to recognise, and made suggestions for modifications.

- **navigation when anchors are shown**

As already mentioned in Chapter 5.4., the 3D viewer allows anchor highlighting to be turned on or off with a window button. When anchors *are* highlighted, navigation could either still be possible or else a simple mouse click (without SHIFT) could be used to activate a link.

Almost all test persons preferred to be able to perform motions when the anchors are shown. The reason is that they do not know in advance where the anchors are and so they want to search for them without often switching between two modes. Only a few users preferred pure motion and picking modes, because it allows all operations with a single hand.

Both alternatives are offered by the system (a toggle *enable motion* is on by default), but as time was limited they were not compared in the usability tests.

- **anchors (general)**

- The term "anchor" is not necessarily familiar to users.
  "Information" or "cross reference" would be better understood.

- ”When moving the mouse over an anchor the object could be highlighted (or the mouse cursor could change its shape).“
  Beside the computational overhead, the user would have to scan the scene for anchors (see general considerations for anchor highlightings in Chapter 5.4.). It would also be disturbing during motion.

- **mode "bounding cube"**

- ”The bounding cubes are not suited for large thin objects like tables and chairs, where the wire cube encloses large non clickable areas.“

- ”When there are many objects in the scene, it is hard to see to which object a bounding cube belongs.“

- **mode "brightness"**

- ”A high contrast in brightness values is necessary for a clear recognition of anchors.“

- ”*Brightness* could be combined with *colour code*: brightly coloured anchors and dark, grey non-anchors.“

- **mode "colour code"**

- ”Dark anchors are rather hard to recognise, e.g. black objects become dark brown. Would it be better to assign the same brightness to all anchors?“
  In that case an anchor in front of another one could be hard to see. It would also entail a loss of realism.

- **mode "colour code"**

- ”The many edges are disturbing, especially when curved surfaces are approximated with many faces. A highlighted silhouette would be pretty.“

- ”Running dashed lines or blinking lines would be more attractive than solid lines.“

7.4.2. Usability Tests of Anchor Highlighting Modes

For an introduction into the system and to get familiar with navigation a demo scene (a room) was presented to the test persons. Navigation mode *heads-up* was used, since it is easy to learn, users did not need to concentrate on navigation functions and could pay attention to the anchors.

Each anchor highlighting mode was explained briefly in the demo scene, followed by the test itself. Each test was of equal difficulty: Subjects had to walk into a building (one entry for each test), turn either to the left or right side and click on every anchor in the room. The number of anchors found and the number of miss-clicks was recorded as well as any remarks the subjects made.

After these objective tests, other scenes (a model of an office and a model of the IICM institute) were presented to the subject. In this subjective test the test person had free choice of anchor highlighting mode.

Finally, the test persons had to fill in a questionnaire with questions to each mode: how easy it is to recognise anchor objects and how visually attractive the mode is. The users were also asked which movement mode(s) they liked best and some general questions to the program as a whole, like the choice of colours.

7.4.3. General Results

During the usability tests the following observation was made:

Some users tend to ignore *any* visual cue telling which objects lead to further information. When asking why, typical answers were "*yes*, I saw that the table was *not* highlighted, but I expected a link because there was also one in the room before" or "*yes*, the lamp on the table *is* highlighted, but what interesting information could be associated with a lamp?"

This tells that attention should be paid to a careful *choice* and placement of anchors. Remarkable objects are expected to be anchors, mundane objects tend to be ignored. This can be more important than the best anchor highlighting method.

The second fact was due to a setting of the window manager. When a window was newly opened (e.g. for displaying a raster image or text after activating a link) only a window frame appeared, which had to be placed with a mouse click. This feature confused most of the users (in particular there is no such option under MS-windows) and should be avoided.

7.4.4. "Bounding Cubes"

- Many users (mainly experts) expected that they can click anywhere in the bounding cube.

- Picture frames were rather difficult to recognise because the bounding cube degenerates to a flat wire frame.

- Bounding cubes may be clipped by other faces, e.g. the cube of a door by the wall. This results in a strange appearance.

Table 7.11 summarises the quantitative results for mode bounding cubes obtained in the usability tests.

bounding cubes	no. of anchors found	no. of miss-clicks	ease of recognition	visual appearance
novice users				
average	8.75	3.44	4.22	4.69
std. deviation	2.00	4.23	2.76	2.91
expert users				
average	8.33	2.38	5.16	4.06
std. deviation	2.04	2.91	2.25	2.14
all users (average)				
average	8.54	2.91	4.69	4.38
std. deviation	2.02	3.57	2.51	2.53

Table 7.11: Evaluation of *bounding cubes*
[found anchors out of 10, answers 0 to 10 points]

The high number of miss-clicks is due to the fact that only the object itself was allowed to be clicked and not the whole bounding cube. This also explains the high standard deviation. Most test subjects reported that the bounding cubes are difficult to recognise, and difficult to associate with the anchor objects. Also the visual appearance was reported to be poor.

7.4.5. "Brightness"

- A few test persons also thought very dark objects could be anchors because they differ from the other objects.

- Some faces of anchor objects are brighter than others due to lighting effects and shading. Many test subjects thought that they may only click those faces.

- The visual impression was said to be gloomy.

The quantitative results for *brightness* are summarised in Table 7.12.

brightness	no. of anchors found	no. of miss-clicks	ease of recognition	visual appearance
novice users				
average	8.78	3.94	7.19	6.56
std. deviation	1.49	6.95	2.48	3.05
expert users				
average	9.05	2.00	4.53	5.63
std. deviation	1.42	2.52	2.96	2.72
all users (average)				
average	8.92	2.97	5.86	6.10
std. deviation	1.46	4.74	2.72	2.89

Table 7.12: Evaluation of *brightness*
[found anchors out of 10, answers 0 to 10 points].

Expert users achieved better results using *brightness* than novice users, they found more of the anchors and had less miss-clicks than novice users.

The miss-clicks occurred on two kinds of objects: rather bright non-anchors which were not clearly distinguished from anchors and also very dark objects, "because of their difference". As it can be seen from the high standard deviation of miss-clicks, some novice users clicked everywhere because they were not sure. On the other hand novice users find that anchors are easy to recognise in mode *brightness*; the reason for this divergence is not clear. Expert users told that finding anchors is difficult in mode *brightness*.

7.4.6. "Colour Code"

- The grey appearance of non-anchors was not disturbing for most users.

Table 7.13 shows the quantitative results obtained for *colour code*.

colour code	no. of anchors found	no. of miss-clicks	ease of recognition	visual appearance
novice users				
average	9.79	1.88	7.97	6.88
std. deviation	0.81	4.54	2.96	3.48
expert users				
average	9.48	0.38	7.66	4.38
std. deviation	1.41	0.70	3.12	3.25
all users (average)				
average	9.64	1.13	7.82	5.63
std. deviation	1.11	2.62	3.04	3.37

Table 7.13: Evaluation *colour code*
[found anchors out of 10, answers 0 to 10 points].

Remarkable is the high number of found anchors and the low number of miss-clicks. This tells that colour code is very effective to use. The - very few - miss-clicks occurred mainly on rather dark objects. The users were not sure if they are dark grey or dark brown.

Also the mark for the recognition of anchors was very high. The visual appearance was graded differently by novice and expert users. Novice users found this mode very pretty whereas expert users found the grey scale picture unappealing.

7.4.7. "Colour Edges"

- If an anchor object consists of many small parts, those parts may be misunderstood as separate anchors, for example doors with handles.
  This problem does not arise with the other modes.

- Colour edges is more appealing than brightness (but only if the object does not have many edges), but more difficult to recognise (for example picture frames with few edges).

The quantitative results can be found in Table 7.14.

colour edges	no. of anchors found	no. of miss-clicks	ease of recognition	visual appearance
novice users				
average	7.92	1.63	5.13	5.31
std. deviation	2.16	4.04	3.17	2.63
expert users				
average	8.33	0.69	5.94	7.50
std. deviation	2.12	0.85	3.52	2.50
all users (average)				
average	8.13	1.16	5.63	6.41
std. deviation	2.14	2.45	3.35	2.57

Table 7.14: evaluation *colour edges*
[found anchors out of 10, answers 0 to 10 points].

Colour edges, both objectively and subjectively, was more appealing to expert users than novice users. Expert users found more anchors and made fewer miss-clicks. They also found the anchors easier to recognise and gave a significantly higher grade for visual appearance than novice users.

Some (but few) miss-clicks were due to flat objects with few edges like picture frames. The lines were not seen as a form of highlighting but as part of the object.

7.4.8. Comparison of Modes

The first table summarises the number of anchors found in each mode. For simple comparison the numbers have been scaled into a range of 10 possible anchors.

number of anchors found	bounding cubes	brightness	colour code	colour edges
novice	8.75	8.78	9.79	7.92
expert	8.33	9.05	9.48	8.33
average	8.54	8.92	9.64	8.13

Table 7.15: Number of Anchors Found [out of 10].

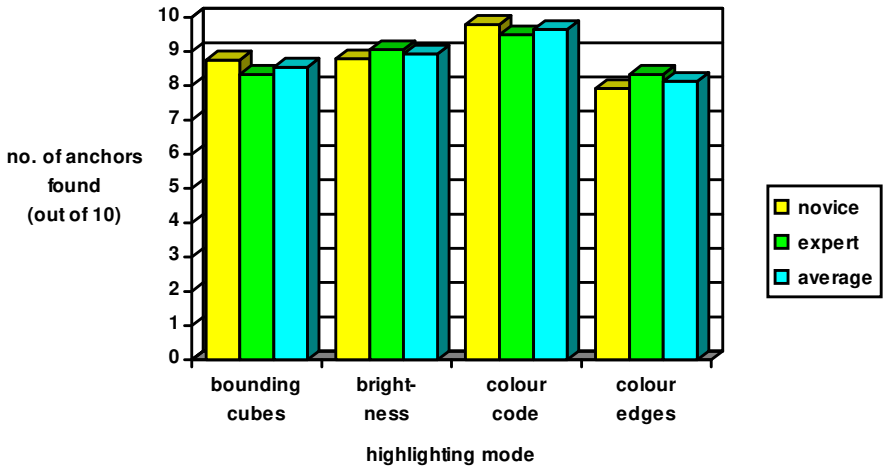

Figure 7.6: Number of Anchors Found

As Table 7.15 and Figure 7.6 show, both novice and expert users found most of the anchors with mode *colour code* and the fewest number with *colour edges*.

The next table summarises how often test persons clicked objects thought to be anchors, but which were actually not anchors.

number of miss-clicks	bounding cubes	brightness	colour code	colour edges
novice	3.44	3.94	1.88	1.63
expert	2.38	2.00	0.38	0.69
average	2.91	2.97	1.13	1.16

Table 7.16: Number of Miss-Clicks

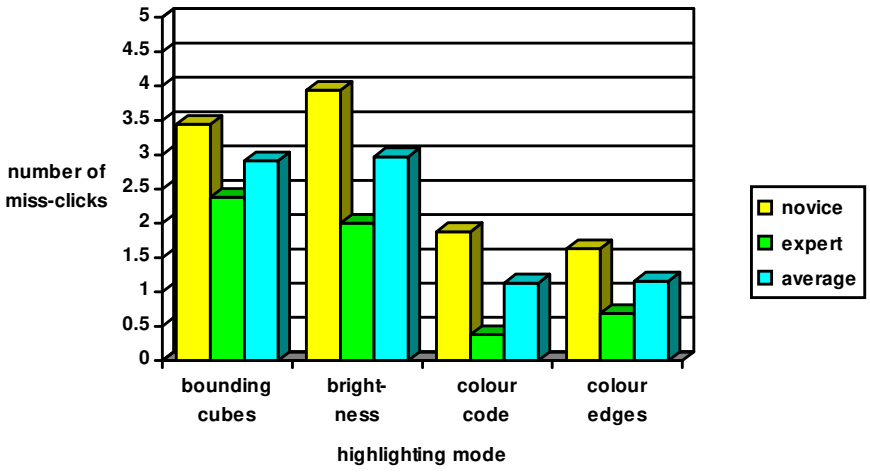

Figure 7.7: Number of Miss-Clicks

A significant difference between novice and expert users is obvious from Table 7.16 and Figure 7.7. Expert users click well-aimed at the highlighted objects, whereas novice users tend to click all around. It can also be seen that the results were significantly better with *colour code* and *colour edges* than with *brightness* and *bounding cubes*.

Interesting is the combination of the previous two statistics. *Colour code* is most effective to use: it had the highest number of detected anchors and the lowest number of miss-clicks. With mode *brightness* subjects tended to click everywhere: many anchors were found, but the number of miss-clicks is quite high. The opposite occurred with *colour edges*: anchors tend to be overseen, but only few non-anchors were clicked. *Bounding cube* had rather poor results in both categories.

The following two tables were derived from the evaluation of the questionnaires. They concern how easy anchors are to recognise and the visual appearance of the anchor highlighting modes.

ease of recognition	bounding cubes	brightness	colour code	colour edges
novice	4.22	7.19	7.97	5.31
expert	5.16	4.53	7.66	5.94
average	4.69	5.86	7.82	5.63

Table 7.17: Ease of Recognition of Anchors [0 to 10 points].

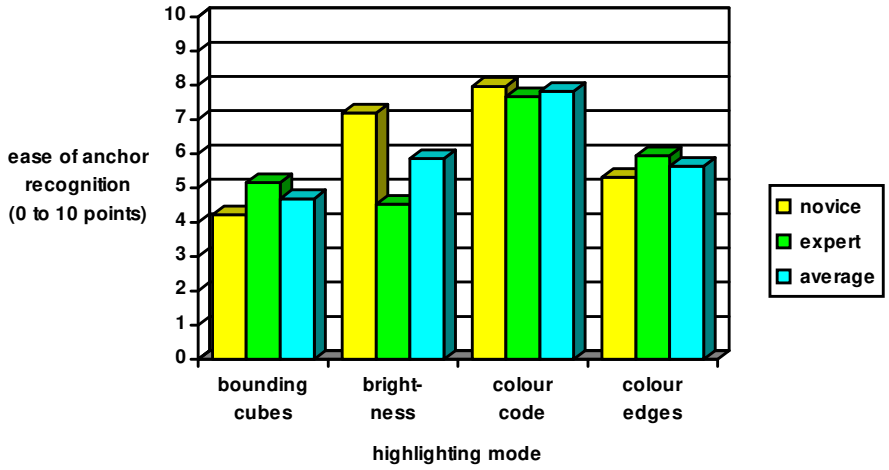

Figure 7.8: Ease of Anchor Recognition

As Table 7.17 and Figure 7.8 show, novice and expert users agree that anchors can be most easily recognised with *colour code*. Concerning the other two modes their opinions differ: novice users also favourite *brightness* and *colour edges* whereas expert users favourite *colour edges* and *bounding cubes*.

visual appearance	bounding cubes	brightness	colour code	colour edges
novice	4.69	6.56	6.88	5.31
expert	4.06	5.63	4.38	7.50
average	4.38	6.10	5.63	6.41

Table 7.18: Visual Appearance of Anchor Highlighting [0 to 10 points].

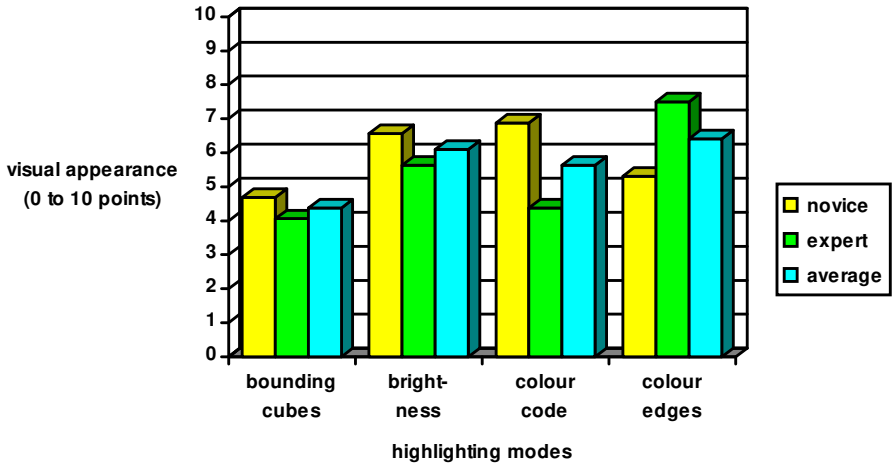

Figure 7.9: Visual Appearance of Anchor Highlighting Modes

Here the opinions of novice and expert users differ very much, as Table 7.18 and Figure 7.9 show. Novice users find *colour code* and *brightness* most appealing, followed by *colour edges*. Expert users were most satisfied with *colour edges* and *brightness*, followed by *colour code*. Both user groups agreed that *bounding cubes* are not very appealing.

In the questionnaire test persons were asked for their favourite mode of anchor highlighting - the mode they liked best and which they would use if free to choose. The results are summarised in Table 7.19 and Figure 7.10.

favourite mode	bounding cubes	brightness	colour code	colour edges
novice	0.0	12.5	75.0	12.5
expert	0.0	7.1	50.0	42.9
average	0.0	9.8	62.5	27.7

Table 7.19: Favourite Anchor Highlighting Mode [in percent].

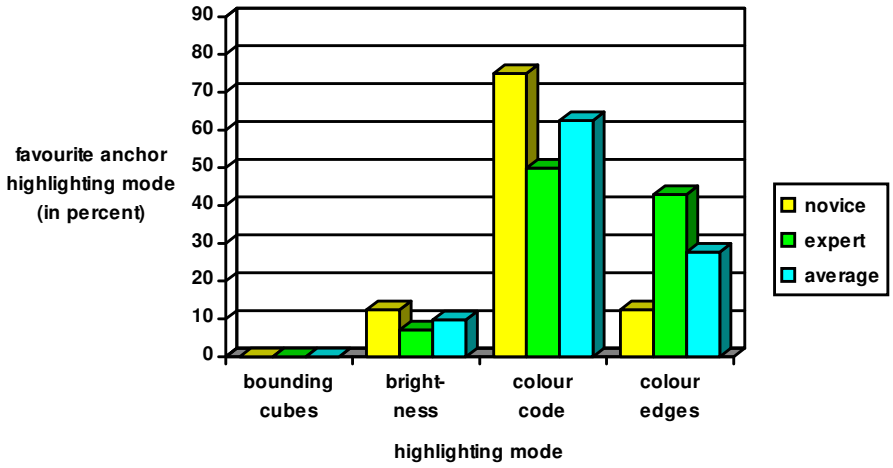

Figure 7.10: Favourite Anchor Highlighting Mode

For novice users the choice is clear: three out of four users prefer *colour code*. Also half of the expert users favourite *colour code*, but also almost half prefers *colour edges*. *Brightness* is liked best by about one tenth of the users, and none of the 32 test persons preferred *bounding cubes*.

Finally, Table 7.20 shows the ranking of the modes for anchor highlighting obtained from the tables above. For a better overview it does not distinguish between novice and expert users. The last line was obtained by combining the ranking of found anchors, miss-clicks and the results of the two questions.

ranking	bounding cubes	brightness	colour code	colour edges
no. of a. found	3	2	1	4
no. miss-clicks	3	4	1	2
ease of recogn.	4	2	1	3
visual appear.	4	2	3	1
favourite mode	4	3	1	2
overall	4	3	1	2

Table 7.20: Ranking of Anchor Modes [1 = best].

7.5. General Results

Some questions on the questionnaire were not specific to either navigation or anchors but concerned usage of the 3D viewer as a whole.

- What did you miss in the program?

- a help system and an explanation of mouse button functions.
- another input device other than a mouse (joy stick, space ball, etc.).
- automatic collision preventing (walking/flying through walls).
- orientation help.
- adjustable mouse sensitivity/speed.

- What impressed you most?

- the range of possibilities for navigation in 3D space.
- speed of the program and graphics.
- interactivity, no limitations of views.
- *heads-up* and *fly to*.
- nothing - or nothing special.

- If you would further use the program, what would you do next?

- use the navigation modes I had difficulties with (mainly said by novice users).
- use the navigation modes I like best (mainly said by expert users).
- view other scenes, further explore the model of the IICM institute.
- try several other menu options.
- don't know - the test environment was too boring!

8. Conclusions

Hypermedia is becoming a more and more widely-used tool for easy access to large amounts of data. The usage of different media increases ease of understanding. Links allow browsing through large amounts of data, without being bound to a predetermined linear path.

Hyper-G - a modular, large-scale, general purpose, networked hypermedia system - is an ambitious undertaking in hypermedia with many innovative features and a wide range of applications. It is designed to run on a variety of hardware platforms and allows access to heterogeneous information.

The Hyper-G *3D Viewer* is an adaptive program, offering navigation techniques for many kinds of 3D scenes. An important aspect in design was the ease of use, such that also novice users feel safe when navigating in 3D space. The 3D Viewer integrates well into Hyper-G and further enlarges the possibilities of hypermedia. With increasing computer performance it will be possible to create highly realistic 3D models of data that can be viewed in real time. A wide range of possible applications exists: from technical illustrations for manuals to models of buildings, from touristic guides to 3D representatives of hypermedia networks.

Five *navigation* modes were implemented. The first, *flip object*, is an object movement technique, suitable when the whole scene represents a model of one object. The mouse movements result in object manipulations (translation, rotation, and zooming). The second mode, *walk*, has been designed to be effective for frequent usage. The usability tests proved that this mode allows fast navigation once the interface has been learned. The aim of *fly* is to allow continuous motion without dragging the mouse, which some users are familiar with from flight simulators. *Heads-up* intends to minimise memory load, icons on the window give the user a feeling of security and offer an intuitive interface. *Fly to* for point of interest-movement is not sufficient as a mode of its own, but a useful addition to *flip object* and *heads-up*.

For highlighting source *anchors* four different modes have been implemented. *Bounding cubes* encloses each anchor object with a cube. The cubes are rather hard to associate with the corresponding objects and their visual appearance is poor. Mode *brightness* renders anchor objects bright and non-anchors dark. The contrast between bright and dark objects has to be enough for clear anchor recognition, but thus resulting in a visually unappealing scene. In mode *colour code*, the edges of anchor objects are highlighted with a different colour, which is striking but sometimes makes recognition of smaller objects difficult. Probably the best approach is mode *colour code*. Here anchors objects are drawn entirely in a selected colour (with their natural shading) whereas non-anchors are drawn in a shade of grey. The usability tests proved that *colour code* had good visual appearance and clear recognition of anchors.

The *usability tests* have shown that the 3D Viewer can be used well by novice users and is also attractive to expert users. They further underlined that there does not exist a "best"