The Harmony Information Landscape: Interactive, Three-Dimensional Navigation Through an Information Space

Diplomarbeit in Telematik

Martin Eyl

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

Fertigstellung:	18.10.1995
Prüfungsfach:	Informationsverarbeitung
Betreuer:	Dipl.-Ing. Keith Andrews
Begutachter:	O. Univ.-Prof. Dr. Dr. h.c. Hermann A. Maurer

Acknowledgments

I would to thank a number of people at the IICM for their support during my work and preparation of this thesis:
Jürgen Schipflinger
Dipl.-Ing. Michael Pichler
Dipl.-Ing. Keith Andrews
Univ.-Prof. Dr. Hermann A. Maurer

Abstract

Computer have made it possible to store and process larger and larger amounts of information, but humans have problems to manage such large amounts of data. Hence user interfaces have improved and the use of spatial metaphors and hyperlinks and the 3D representation of information have gained in significance. This thesis first discusses a number of research projects which use spatial metaphors and then introduces you to hypertext and the Harmony Information Landscape which visualizes the hierarchical structure of information as an information landscape. This information is stored on a Hyper-G server which is an Internet-based, large-scale hypermedia system. Blocks representing collections of individual pieces of information are spread out on a plane and one can interactively fly over this virtual landscape. Several different navigation modes are provided.

Contents

Chapter 1

Introduction

Nowadays it is possible to view pictures of Vienna, to access a database in Sydney or to listen to the voice of the president of the USA without leaving one's chair in front of the computer. Increasing computer performance and the increasing capacity of mass storages (hard disk or CD-ROM) make storing and managing of large amounts of data possible. In a network the capacity is multiplied by the number of computers which are connected to one's computer. The Internet nearly covers the whole world and provides the largest database mankind has ever had. The number of databases and information systems and the net itself grow and grow. And so the problem is not that the desired information is not available but the problem is to find this information. These large amounts of data make tough demands on the interface between the computer and the user. The data will be and are already available to many people who are not computer experts. The ultimate goal is that everybody should be able to use an information system and to find the desired information without any specific knowledge. On the other hand, the new and powerful computers can also be used to build very powerful user interfaces and increasing computer performance makes new aids for the user possible. The Harmony Information Landscape has been developed to provide such a powerful interface.

Hypermedia is a possibility to organize large amounts of heterogeneous data like texts, images, digital audio clips and video clips or 3-dimensional objects and scenes. The data are divided in arbitrarily large pieces called documents or nodes (for example a picture or a page of text) which are connected to each other by links. The links show readers a relationship between the documents and links can be followed (e.g. by a mouse click) to its destination document which could be for example more detailed information or an animation which illustrates a text. The nodes and links form a web of information in which one can freely move around and one can always access related information by activating a link. But hypertext also presents new usability problems of its own. Comparable to the web of streets in a town, the user can get lost and therefore several navigation aids have been developed like backtracking, history list, searching, browsers, overview maps, metaphors or a hierarchical structure.

Hyper-G is a general-purpose, large-scale, distributed, multi-user hypermedia information system, which is based on the server-client model. It is the second generation of Internet-based hypermedia systems and has been developed at IICM in Graz. Hyper-G provides an additional structure beside links. Collections (of documents) and clusters build up a hierarchical structure. Harmony is the Hyper-G client for X Windows on UNIX platforms and it provides several different navigation aids, among others the Information Landscape. The Landscape is part of Harmony and presents the collection hierarchy in an open 3-dimensional landscape.

Spatial metaphors can be used to present large amounts of information in a more compact and natural way. We live in a 3-dimensional world and we are used to move in this world and to handle with 3-dimensional objects. This existing knowledge and natural abilities should and can be used for an interface. Screen space is limited and one can use 3D visualization to maximize effective use of this screen space. This thesis introduces several projects using spatial metaphors and 3D presentation.

The Harmony Information Landscape presents the collection hierarchy of the Internet-based Hyper-G servers but it does not portray the links of the hypermedia system. The collection hierarchy is an hierarchical structure and can be used to find one's way in the web of links. Thus the Landscape is an additional navigation aid for the user.

The Landscape is a first attempt in Hyper-G to take advantage of a spatial metaphor. The collection hierarchy is presented as a tree in an arbitrarily large landscape. Collections, clusters and documents are represented by blocks. The document blocks are placed upon the collection socles and cluster socles and titles are displayed in the front of the blocks. The colour of a block shows the type of the document and the size of a document is mapped to the height of the block. In the future one will also use other properties (for example the shape) to represent an attribute. Users can open and close collections and view documents by double-clicking with the left mouse button. A second window provides an overview map of the current collection tree. The landscape can be explored interactively. *Interactive navigation* gives the impression of moving in a real 3-dimensional landscape and the drawings are made in real time. Several different navigation modes have been implemented. The system also provides smooth animation to take the user to a desired object (for instance to an object which the user has chosen from the result list of a search query or from the history list). A fisheye view means that only objects near the viewpoint are displayed in great detail and it is used to avoid unnecessary drawing (for example a very small title far away).

Chapter 7 should be read if the user wants to work with the Landscape the first time.

Chapter 2

Hypermedia and Information Systems

In 1945 already Vannevar Bush published a manuscript about an information system which he had developed in order to store and access information [8]. This system, which was never implemented, was the first description of an hypertext system. At that time the explosion of scientific information was already the main motivation for this system. Today the situation becomes worse and worse and with the possibilities of a computer network the amount of available information increases enormously. So hypertext is a very important tool to manage this information and hypertext can now be implemented with commercially used technology.

In the first part of this chapter, I am going to explain the facilities of hypertext, problems, and possible solutions of these problems. In the second part I am going to introduce you to distributed hypertext and information systems spreaded out over the Internet. Distributed means that the system is not only located on a single computer but on many computers in a network. The Internet covers nearly the whole world and it supports access to thousand of databases and information systems.

2.1 What is Hypertext / Hypermedia ?

Till now we are used to read a text sequentially [21, 5, 22]. We start to read a book at the beginning and we finish at the end. Hypertext is non-sequential. There is no fixed sequence but the user can decide where to read further (see Figure 2.1). The author of the text arranges several different possibilities in advance. Comparable to footnotes in a printed text the reader determines to read it or not.

The nuclear elements of hypertext are nodes and links. The content of a node could be a single word or a long text. A node could also contain other forms of data like a picture, a sound, a video and so on. In that case hypertext is called hypermedia[1]. Nodes are connected to each other by links. Normally a link is directed from a source anchor to a destination anchor. The anchor of a link can be associated with a small part of the node (for example a word) or with the whole node. The source anchor of a link is

[1]Hypertext systems have some advantages over a hypermedia system for example economy, ease of development and ability to run on cheaper and less powerful computers.

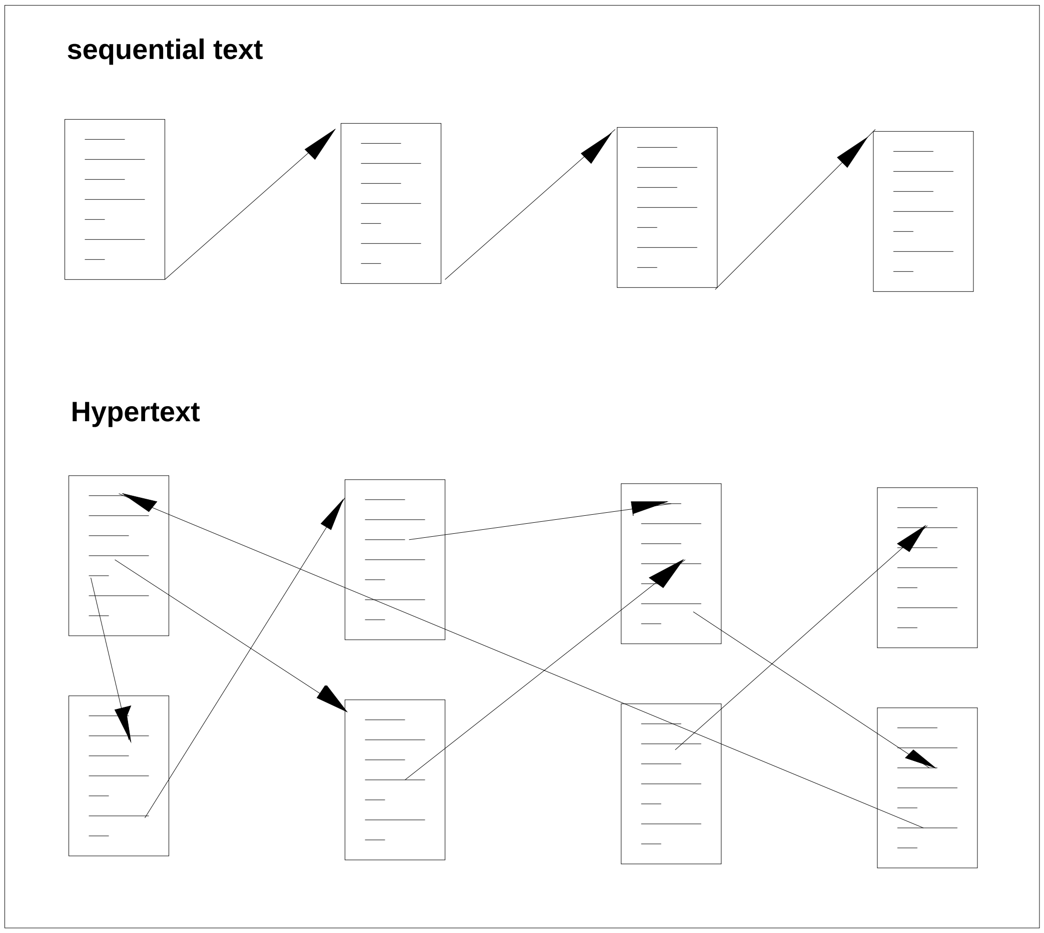


Figure 2.1: Sequential Text and Hypertext

displayed as an anchor icon for example a button or a word with a different colour. Now the user can activate a link (for example by a mouse-click) and the system will present the destination anchor. Authors of a hypertext can place links to show readers a relationship between a source and a target node or the hypertext system itself can generate links automatically. In some systems the links is of a type to characterize a relationship, for example a definition link or an author link. And so the nodes and the links make up an information web in which the reader can freely move around.

For example the readers select an article about a particular bird. Then they can determine to get more information about the propagation, the food or the appearance of the bird. A click on the word "propagation" with the mouse opens another window which shows a map of the world with coloured regions. The readers can also click on the word "appearance" to look a video. Clicking on the bird in the video starts a song of this bird and so on. The readers can explore the information corpus as they like and so the users examine only the interesting parts.

The base of hypertext is a database with additional features. The items of this database can be completely different for example a picture, a text or a video. The links can be embedded in the text (WWW) or they are also stored in the database. The database can be local on a computer or it can be distributed in a network.

2.2 Application of Hypertext

Hypertext makes new demands on reader and authors. Hypertext readers must assume a much more active role than readers of printed text. Hypertext authors must organize the information into numerous fragments or nodes and connect them with meaningful links. Not all information is divisible in fragments and therefore not all application are suitable for hypertext. Here are few possible applications:

1. Documentation
   A online documentation in hypertext form on a computer has some important advantages. The documentation is always available. When a problem occurs, one can use it immediately without looking for a book. The user doesn't have to read the whole documentation to solve the problem. One only needs a small part with the respective explanation. Hypertext is a very usable tool to access a particular part of a text and so the user can easily find the sections that interest him or her. Many software packages have been delivered with online manuals in hypertext form. An example is the documentation of Borland C++ compiler. Explanations of commands are connected by links with program examples or with other commands and each written word in the editor is an anchor of a link. If the written word is a command, then the destination is the explanation. The destination of a link from a unknown word is a list of similar commands. Another possible application besides computer manuals is a repair manual which contains descriptions of how to repair cars or bicycles linked to video clips. The documentation and the repair manual of an aeroplane has millions of pages and finding information about a small component can be improved by hypertext.

2. Advertising
   Hypertext with multimedia is an excellent possibility to introduce a product to a customer. A hypermedia product catalog can provide not only text and pictures to attract the attention of the people. For example a car catalog can contain a driving simulation. One problem of printed catalogs is the right amount of information. Some people feel bored with a long text. Other customers want more information about a particular product. In a hypermedia catalog the customer can choose a product and can access the information that interest him or her. The companies can spread their trade shows or product catalogs with the help of the Internet, disks or CD-ROMs.

3. Medicine
   Hypertext is a kind of database and so it can be used to store different types of data. In a hospital a patient file contains medical check-up results, diagnosis, X-ray picture, cardiac sounds and the like [14]. At the present one part of the data is stored on the computer and the other part exists on paper and film. The goal is to store the whole patient file on the computer and to use a hypermedia system. A patient can be treated from several doctors and all doctors can access the information by using a local network. There can be links inside the file and between files.

4. Education
   The information stored in a hypertext system can also be knowledge and a student can use hypertext for learning. A encyclopedia or a dictionary is a kind of printed hypertext and can also be implemented on a computer. Every word in a text, which is written in a foreign language, can be linked to a translation of the word. Hypertext also enables user to view parallel the original version and the translation of the text. Another possibility to learn a foreign language is to show film clips of conversation. If the students do not understand the conversation, they can use the hypertext facilities and link to a film where the people speak more slowly. Another educational hypertext is the museum information system where the users can look round and view the museums objects that interest them.

2.3 Navigation in Hypertext

Hypertext has much to offer in terms of managing large amounts of information but it may also present a new set of usability problems of its own. The main advantage of hypertext is the possibility to travel about the web of information. This advantage is simultaneously a new challenge for the readers. They must be able to find their way in the hypertext. Most hypertext systems provide some navigation aids to help the users.

2.3.1 From Wandering to Searching

Browsing is to move from node to node by following links. The way to browse a hypertext depends on the task or the intention of the reader. For instance one can look for a particular chunk of information to solve an existing problem. Another possibility is that a user wants to get information about a common topic. Or a user has only a vague idea and scannes the hypertext to get a sense of other related concepts. Browsing can range from curiosity about any or all elements to a constrained search. Smith el al [29] distinguish between five methods of exploring or browsing a hypertext :

- scanning - covering a large area without depth to get an overview
- browsing - following a path, which is a train of thought, until goal achieved
- searching - striving to find a explicit goal
- exploring - finding out the extent of the information given to get an overview
- wandering - purposeless and unstructured globe-trotting

Smith asserts that wandering has no place in hypertext but I think it can be quite amusing and entertainment has its place in hypertext.

2.3.2 Lost in Hyperspace

The most commonly cited problem in the hypertext literature is that of disorientation or becoming "lost in hyperspace" during browsing. This problem grows with the size of the hypertext comparable to a town. In a small village one can easily maintain orientation but in a big town it is almost impossible without navigation aids. The web of nodes connected by links can be very complicated and confusing. Here are some possible user statements:

- I don't know where I am in the information space.
- I don't know how to access something I belive exists there.
- I don't know how to get back to where I came from.
- I am not able to find something again across which I stumbled.
- I forget which nodes I have been visited.
- I neglect to return from a digression.
- I find interesting sidetracks which distracts my attention from the main task.
- I am not able to form a coherent understanding of what I have been read.
- Is there more or any information that interest me ?

The origin of such problems is often the unfamiliarity with the structure or organization of the network. The users don't know the extent of the network or all relevant nodes. Perhaps some parts of the network are never explored. Another group of problems is called cognitive overhead. The users have to find their way through the hypertext and they can lose track of the main task because there is interesting information on the way. But this information can also help to concretize or redefine the goal.

An origin can also be inexperience of the reader. Hypertext is a new information medium and people are only used to read books. And so they must learn to work with a hypertext system and to find their way. The author is often responsible for some problems. Ill-organized and unkempt writing in a book is very confusing [6]. In a hypertext system such authors produce obscure, confusing and frustrating information webs. The correct partition of the information and a meaningful linkage is an important and very difficult task for the author. Not only new documents but also existing documents have to be transformed into a good hypertext.

2.3.3 Navigation Aids

Several possible aids and solutions to these problems have been developed. A very simple kind of help for the user is the guided tour which leads the reader through the hypertext. Hypertext systems can provide different guided tours with particular topics and can also suggest additional nodes with already visited topics. It can be very helpful for beginners. But guided tours are like a sequential linear form of information and the advantage of hypertext, which is the open exploratory information space, is not used.

Backtracking is a very efficient solution to the problem of getting lost in hyperspace. The backtrack feature allows the reader to get back step by step to the starting point. Users can dare to go out to unexplored places because they can find their way back to a familiar territory by backtracking. If the readers stumble about interesting nodes they can easily find it later. A comfortable form of backtracking is the history list, which allows the user to access any previously visited node immediately. But a history list grows with time and can become very complex and difficult. Another similar tool for the user is a bookmark which is a special form of a history list. The users can decide which nodes are stored in a list and they can revisit them immediately by selecting them from the list. The list and its size is in control of the user.

Indexes are often used in books and can also be implemented in hypertext systems. Users, who are interested in looking up or finding a specific item, can use a index. Indexes can be implemented as a list of links.

Browser and Overview Map

There are parallels between the real world and hypertext. And so the results of spatial cognition research can be applied to hypertext, for example research on mental or cognitive map [28]. A cognitive map is an "orienting schema" or a mental representation of the information web and it is very important for maintaining the orientation. Users develop a cognitive map by interacting with the hypertext as well as via using maps or browsers of the network. A 2- or 3-dimensional browser should possess some properties helping users to acquire a cognitive map. Inhabitants sketched maps of cities, which reveal some of the content of cognitive maps consist of landmarks, paths, nodes, districts and boundaries. The possibilities of districts and boundaries are often not used in browsers. Shum [28] described another important property: "It is important for a node's location to carry information about the content, or spatial organization is meaningless." For example system-readable attributes can be mapped to an absolute location in space. Nodes can be arranged in space in terms of general and detailed information or simple and advanced information. It is also important that the position of the node doesn't change, for instance if new views are computed. We would be confused if things constantly change.

The information web is normally too large for showing all nodes and links on a single map. There is not enough space on the computer screen to show all details. A solution is to show different maps with different levels of detail. Another possibility is the fisheye view [10, 12] which means that parts of the information nearby the current position of the user are shown with great detail. The parts further away, which are not so interesting for the user, are displayed with less detail. A fisheye view is only suitable for highly structured hypertext.

An advantage of 3-dimensional browsers or maps is a simplifying the network structure by eliminating intersection links. The user's position and movement, the size and kind of the information or the number of visits can also be displayed in the overview map.

Searching

Normally an unfamiliar hypertext is too large to find a particular topic by browsing. Many hypertext systems provide a searching feature. A query in a full text search finds occurrences of words specified by the user in all nodes. These nodes can be displayed in a list and the user can immediately jump to a node by selecting it from the list. The list can be sorted by title, author and so on. The search result can also be displayed in an overview map where the matching documents change their colour. The search can be limited to particular parts of the hypertext.

Metaphors

The advantage of a metaphor is that user's existing knowledge and natural abilities are used. The computer user, who wanted to use a database, used to have to be an expert prior but today people should be able to use a hypertext system without any specific knowledge. All people are very familiar with navigation in a 3-dimensional space, a house with certain rooms, objects and so on. An interface can use such everyday spatial environments. And so users instinctively know what they have to do and they can draw on existing world knowledge to act on the electronic domain. Several interfaces using a metaphor have been developed.

Cornell University's Interactive Multimedia Group developed the Bughouse [13], which contains information on the subject of art, history, entomology and so on. They implemented an interactive touch-screen and the readers can just point to the place where they want to go and in this way they can examine the whole house. General themes are organized by rooms. For example, the topic music is placed in the music room. The user can touch objects on the screen to get more information for example a cookbook in the kitchen for the topic food.

A very similar project is an academic department, which was developed by Smith and Wilson [29]. Information about this department is placed in a 3-dimensional model of the building. Information about members and their research interests is assigned to their individual offices, information about facilities is placed in a laboratory and so on. Users can also use a map of the building to find their way.

Waterworth and Singh [32] used islands as metaphor. Archipelagoes, islands and buildings represent a hierarchical information system. An archipelago contains a major class of information and an island a subclass of information and a building contains a topic. Users can explore this world with a vehicle which provide a public and private view and they can decide what they want to see in the window with the private view.

The natural hierarchy of an object (an element is a composition of other elements) is used in a hypermedia system, which was developed by Serra et al[27]. The system can especially be used to provide information about 3-dimensional objects. The base of the hypermedia system is a 3D model. The users can get detailed information by clicking on the part of the 3D model which interest them. Text, image and video nodes have been used and realistic animations have been implemented. Possible applications range from technical objects (for instance an engine or an aeroplane) to an human body.

Hierarchical Structure

As was mentioned before, some hypertext systems use a hierarchical structure. Very large hypertext systems need an additional structure beside hyperlinks otherwise it is almost impossible to get an overview and to find one's way. Gopher and Hyper-G use a kind of hierarchical structure.

Parts of the information web which stand for a class of information can be collect together (in Hyper-G the parts are called collections and in Gopher menus). Several classes can build up another class of information and so on. This hierarchical structure is also a kind of fisheye view. The users can move along the hierarchy to get a detailed view of a part of the information corpus. Simultaneously they have also an overview of the rest.

2.4 Information Systems

In this section I am going to introduce some information systems, which make large amounts of information on the Internet available to every Internet user. Gopher, WAIS, WWW, and Hyper-G are the most prominent. All this systems are based on the client server model. The server manages the documents and the client makes the documents available for the user.

2.4.1 The Internet

The Internet has been developed in the past 20 years [16]. It is growing continuously and it covers nearly the whole world. The net has not been built by only one company or university, it has grown out of many local networks (LAN) which are inter-connected. Each company or university is responsible for their own net. All computers which are connected to the Internet must use the same protocol (IP/TCP) otherwise they can not understand one another. The data are divided in small parts and then the parts are sent to the destination computer where they are put together again. The transport-way between two computers can differ for example if a part of the net is broken then the data are sent another way. And so each computer can communicate with each computer and for instance information systems and databases in Japan can also be used in Europe. These are the basic services provided by the Internet:

- E-MAIL - send and receive electronic mail
- FTP - send and receive data (files)
- TELNET - use a remote computer
- NEWS - discussion-groups with particular topics
- access to information systems and databases

2.4.2 WAIS

WAIS stands for Wide Area Information Servers and is used for information retrieval. There are more than 500 WAIS databases on the Internet and one can search for information (articles) in these databases. The main database contains the addresses of all other databases with keywords and is the starting point of a search. The user determines some keywords and WAIS finds matching databases. Then the user can continue the search in those databases. WAIS is based on a full text search and the search result are sorted by a number, whereby the best matching document gets the largest score. There are no links connecting documents.

2.4.3 Gopher

Gopher was originally developed on the University of Minnesota and provides a hierarchical structure (menus) to access data which are stored on servers on the Internet. When the user starts gopher, the main- menu of the home server is displayed on the screen. Gopher manages different types of menu items: sub-menus, text files (or binary files), search services or TELNET. The user can choose a menu item which can be located on the home server or on any other remote server. That makes no difference for the user. So one can search for a particular piece of information in the whole Internet by browsing the menus or using the search facilities. Gopher uses existing services like telnet or FTP, but is not a real hypermedia system because there are no links in the text files.

2.4.4 World Wide Web

The World Wide Web (WWW, W3 or the Web) is a real hypermedia system and links connect documents, which are spread all over the world. WWW was originally developed at CERN in Geneva and it is the first hypermedia system whose component servers are accessible via the Internet. Information is structured in documents and it manages different types of documents: text, image, and audio and film clips and so on. A text document is stored in so-called HTML format and links are an integral part of the document. Several different clients are available on all major platforms: Lynx (a text-only client), Mosaic and Netscape. This clients provide some aids to find one's way in this large information web (for example history list, bookmarks or backtracking). Mosaic or Netscape can also access other Internet services: FTP, TELNET, Gopher, search facilities, and news groups.

Chapter 3

Hyper-G and Harmony

Hyper-G [1] is a distributed hypermedia information system, which combines the best of Gopher, WAIS, and the World Wide Web (WWW) and provides additional features. Harmony is the Hyper-G client for UNIX / X11. In this chapter I am going to introduce you to Hyper-G and Harmony.

3.1 Introduction

The Internet provides the largest information and communication resource mankind has ever had. The databases stored on the computers in this network comprise the largest database in the world. Several Internet information systems have been developed like WAIS, WWW, and Gopher whereby WWW is the most famous. However, these systems have some drawbacks and weaknesses. So Graz University of Technology developed a new hypermedia system: Hyper-G, a second generation hypermedia system [3, 2, 19, 15]. Different clients are available to present the information stored on a Hyper-G server:

- Text only clients (hgtv)
- Amadeus is the native Hyper-G client for MS-Windows on PCs.
- Harmony is the native Hyper-G client for X Windows on UNIX platforms.

The problems typically associated with large hypermedia systems and the weaknesses of other systems have been considered during the design of this system. As was mentioned before disorientation grows with the size of the hypertext. Hypertext systems (containing millions of documents) must provide several different navigation aids and retrieval facilities and the Landscape is one of them. Harmony and Hyper-G relies not only on one kind of aid. The navigation aid used depends on the task or intention of the user.

Another important point is authoring. A single person can not write and maintain a large database. If several authors work in a hypertext system, access rights become a necessity (similar to the UNIX directory). The author should also be able to link comfortably documents even if the author has no write access to the document. The consistency and integrity of data must be guaranteed in a large hypertext (for example avoiding dangling links when document is deleted). The system must be able to access other databases using the Internet or a local network and the system should be interoperable with other database types like WWW or Gopher. The designers of Hyper-G and Harmony considered these requirements and the problems of large hypermedia systems.

3.2 Hyper-G

Hyper-G is a general-purpose, large-scale, distributed, multi-user hypermedia information system, which is based on the server-client model. The client (Harmony) communicates with the server (Hyper-G). Client and server can run on the same computer, on two computers in a local network, or on the Internet (several thousand kilometers apart).

Hyper-G stores and manages hypermedia data and makes this data available to one or more clients. A particular client, however, always talks to the same server during a session (WWW clients are connected with several servers). When the client wants to access documents from other Hyper-G servers, the server which is connected to the client (the local server) fetches it and delivers it to the client. Hence users only work with one large hypertext and the other servers are not visible for them. Each server knows all the information stored in all other server and stores the remote documents which have been fetched in a local cache. When the same or another client (connected with the local server) asks again for those documents, the server must not fetch it from the remote server a second time. It get those documents from its local storage. The transmission-time for a big document within a local network is much smaller than the transmission-time of a document within the Internet across several thousand kilometers. A server should be installed locally in the user's network even if that server is used for nothing much beyond caching.

The connection with only one server has even more advantages. Regarding access rights, the user has to identify to one server only and the client can be kept simple. The Hyper-G client-server protocol is efficient and connection-oriented.

3.2.1 Collection Hierarchy

Hyper-G provides additional hierarchical structuring facilities beside hyperlinks. This structure is an important navigation aid. Defining a search scope and access control without this structure is quite difficult.

The collection hierarchy is best explained recursively: Every document is a member of one or more collections and every collection is also a member of one or more collections except the root collection. The resulting structure is not really a tree because a document or a collection can belong to more than one parent collection. For example the document with the topic "theory of relativity" can belong to the collection "physics" and to the collection "Einstein". A user can move in the collection hierarchy and view the contents of the collections. A collection contains a list of objects: documents, subcollections, and clusters. A cluster is similar to a collection but when a cluster is visited all of its substructures are visited (visualized) too. Hence a cluster is a kind of multimedia document for example a text, an image, and a video clip simultaneously displayed on the screen. It can also be used to support multilingual documents and version control (see also Figure 3.1).

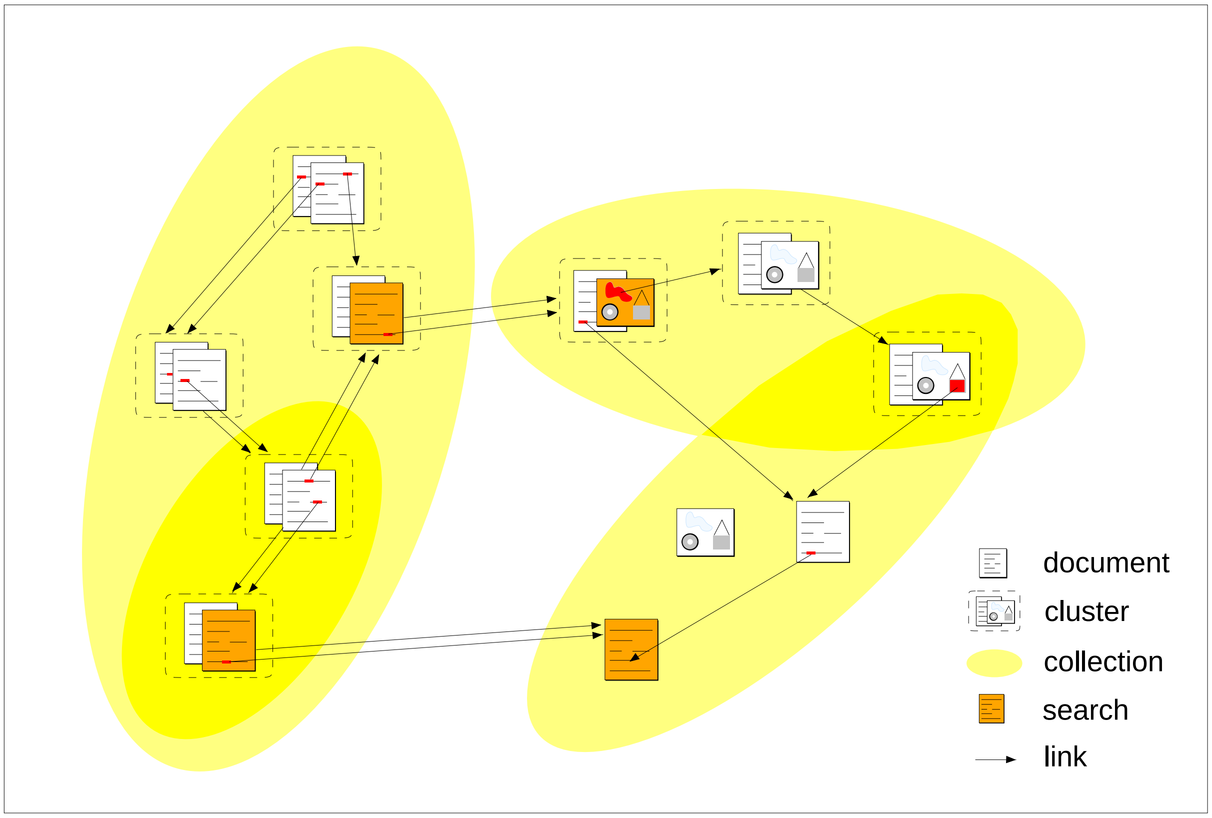


Figure 3.1: The Hyper-G Data Model

In addition and in parallel to other navigation aids the collection hierarchy can be used to get an overview and to orientate oneself in the information space. In this structure the place of a document is obvious and the user can find and find a document again, even in a very large hyperspace. For example a text with the topic “proton” can be placed in the collections “natural science”, “physics” and “atomic physics”. Everybody would look there. Comparable to using maps of the world, of countries, of towns and so on, users can easily get an overview of the content with different levels of detail. Relevant parts of the information universe will not be missed.

3.2.2 Architecture

The Hyper-G server comprises three distinct server processes: full text server, link server and document server (see Figure 3.2). The full text server is not explained further here.

The link server is an object-oriented database of objects and relations between such objects. The objects can be description of documents, links, anchors, collections and so on. The relations are for example which documents belong to which collection or which source anchor are connected to which destination anchor.

The link server assigns object IDs to objects. An object ID is an unique number (similar to an ISBN number). No two objects share the same number. The object IDs of

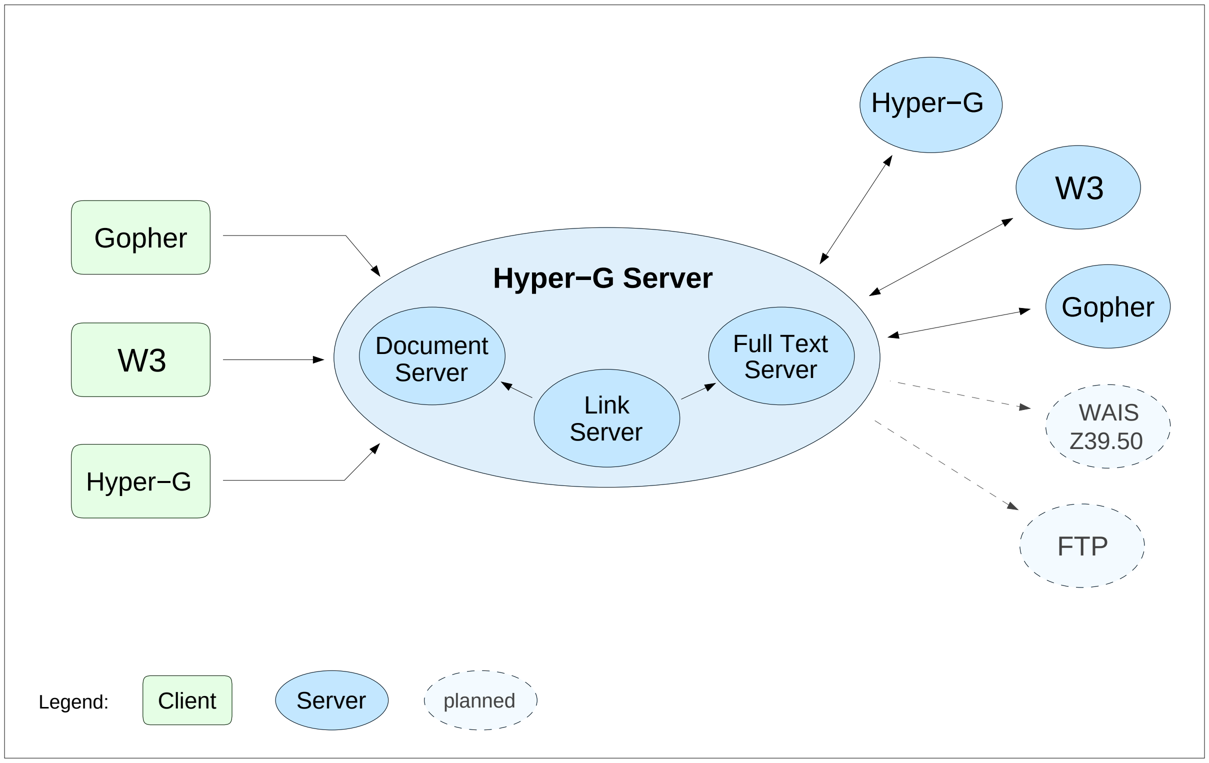


Figure 3.2: Hyper-G Architecture

a deleted object is not used a second time and each version of an object receives a new ID. The link server stores additional information of the objects: title, author, creation date, additional keywords. In the case of documents also the location in the document server is stored which is necessary to retrieve the document from the document server. In WWW links are embedded in documents, in Hyper-G links and documents are strictly separated. That means several advantages over WWW: Accessing only the link-server the user can browse the hypermedia corpus. When the users find a interesting document they can fetch it from the document server. It is not necessary to fetch documents for exploring the collection or the link structure. The link information is stored in the link server, the documents themselves are stored in the document server. Another advantage is the possible attachment of links to read-only documents, for example documents stored on a CD-ROM or on a remote server where the user has no write permission. For example the user can create a link from a dictionary (stored on a CD-ROM) to a private annotation (an additional explanation). Storing links within a separate database has a further advantage: the link server can provide bidirectional links. Usually a link is directed from an anchor node to a destination node. In the case of a bidirectional link one can also follow a link from the destination to an anchor and so one can determine all documents which refer to a particular document. Whenever this document is deleted or modified, documents which refer to the document can be identified and possibly the links can be removed from the web. In a large database it is almost impossible to remember or know all documents connected to a particular document. (Other persons can also link their documents to this document). The link server is able to maintain the consistency and integrity of the information web. It also stores the information about the collection hierarchy and guarantees that every document belongs to least one collection. The link server is also responsible for additional tasks like searching and access control and it can gather detailed statistical data about the system.

As was mentioned before, the document server stores all local documents and caches remote documents. When a document is needed, the document server delivers it to the user. It caches remote documents on local mass storage. This cache memory (for example hundred of megabytes of hard disk) is limited and so the least recently accessed documents must be deleted regularly. In a large hypertext system documents are modified permanently and so the system must guarantee that the user receives always the latest version of a document. A document in the cache can be an old version. A possible solution is that whenever a document is modified, it must be sent again to all relevant caches. A list of all relevant caches is necessary and so on. The solution implemented in Hyper-G is much simpler. Every object or document receives an unique number, the object ID. This means that the new version of a document receives a new number which is passed to the client when the user visits this document. The document server doesn't find a document with the new object ID in the cache and so it fetches the new version from the remote document server. The old version with the old object ID will be deleted because of the least-recently-used strategy of the cache.

3.2.3 Searching

The attributes of a Hyper-G object (such as title, keywords, creation time, expiration time and so on) which are stored in the link server can be used to find documents of current interest. Boolean queries might be for example "Search for all documents with 'theory of relativity' or 'Einstein' in the title", or "Search for all images which have been created yesterday", or "Give me text-documents with 'Windows' in the keywords, created by Smith after 23-05-94". Hyper-G provides also sophisticated full-text search facilities. The result of a search query is a list of matching objects. When the user searches in a large database this list might be very large. In Hyper-G the user can define a search scope in order to reduce the number of matching objects. The scope can be a union of any number of collections, for example a particular set of collections on a single or a number of Hyper-G servers. The search scope can thus range from a single collection on a Hyper-G server to all collections on all Hyper-G servers worldwide.

3.2.4 Access Control

Hyper-G also supports access rights. Users can use Hyper-G anonymously or can identify themselves. Identified persons have their own home collection where they can collect pointers to documents most important for them. Similar to a UNIX file system read and write access to certain parts of the information web (collections) can be granted or denied to certain user groups.

3.2.5 Interoperability

WWW is the first wide-spread hypermedia system on the Internet. A lot of WWW servers are installed in the Internet because WWW server and client are available via Internet and easy to install and clients are available for all major platforms. This large existing pool of information resources should also be available for the Hyper-G user. Thus interoperability of information servers is a very important issue. Hyper-G is able to interact with Gopher and WWW servers and Gopher and WWW clients. When a WWW or Gopher client is connected with Hyper-G server the hierarchical structure must be mapped from Hyper-G to the clients. Gopher provides a menu-tree and so a collection is converted to a menu. Hyperlinks can not be represented in the Gopher metaphor. There is no hierarchical structure in WWW, so the WWW documents are generated to look like menus. Each collection in Hyper-G must be converted to a document containing a menu of links to other documents (sub-menu or sub-collection).

Hyper-G can also connect Gopher and WWW servers. Gopher menus are converted into Hyper-G collections, WWW text documents into Hyper-G text documents. The collections and documents of the WWW and Gopher servers are part of only one collection hierarchy which the Hyper-G client presents and so the user works with one large virtual information space.

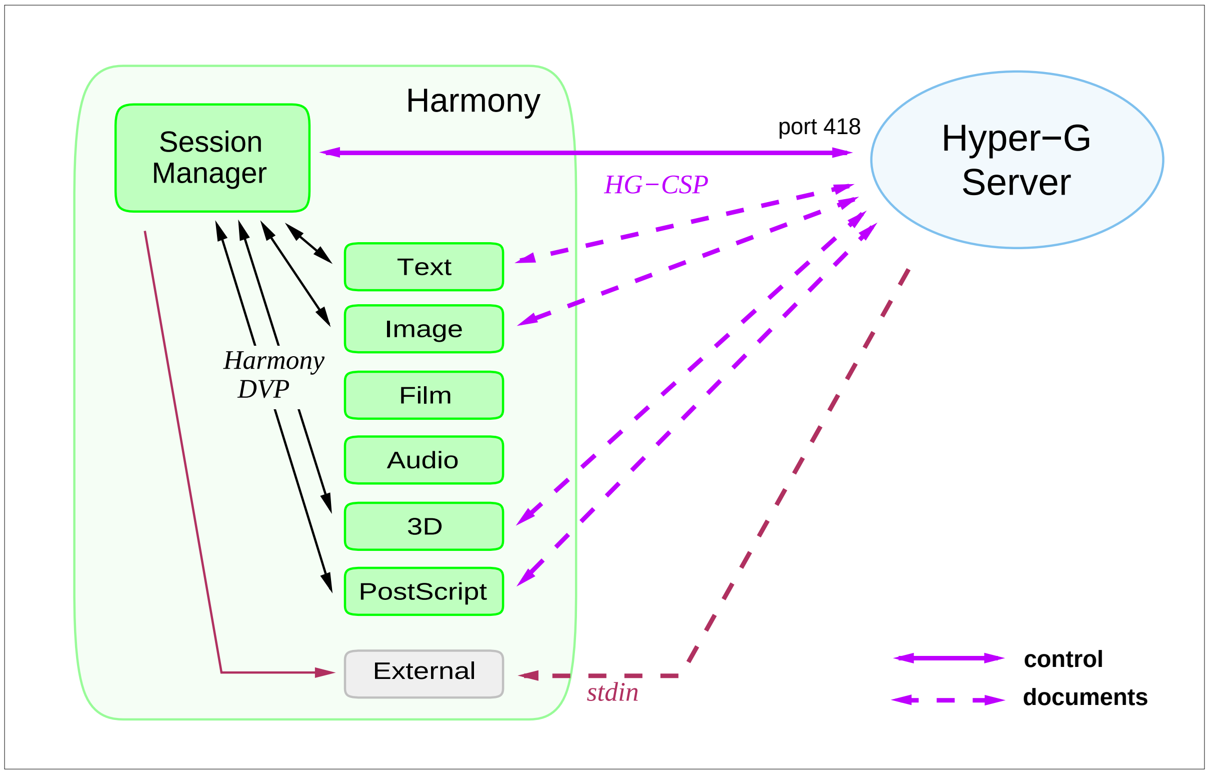


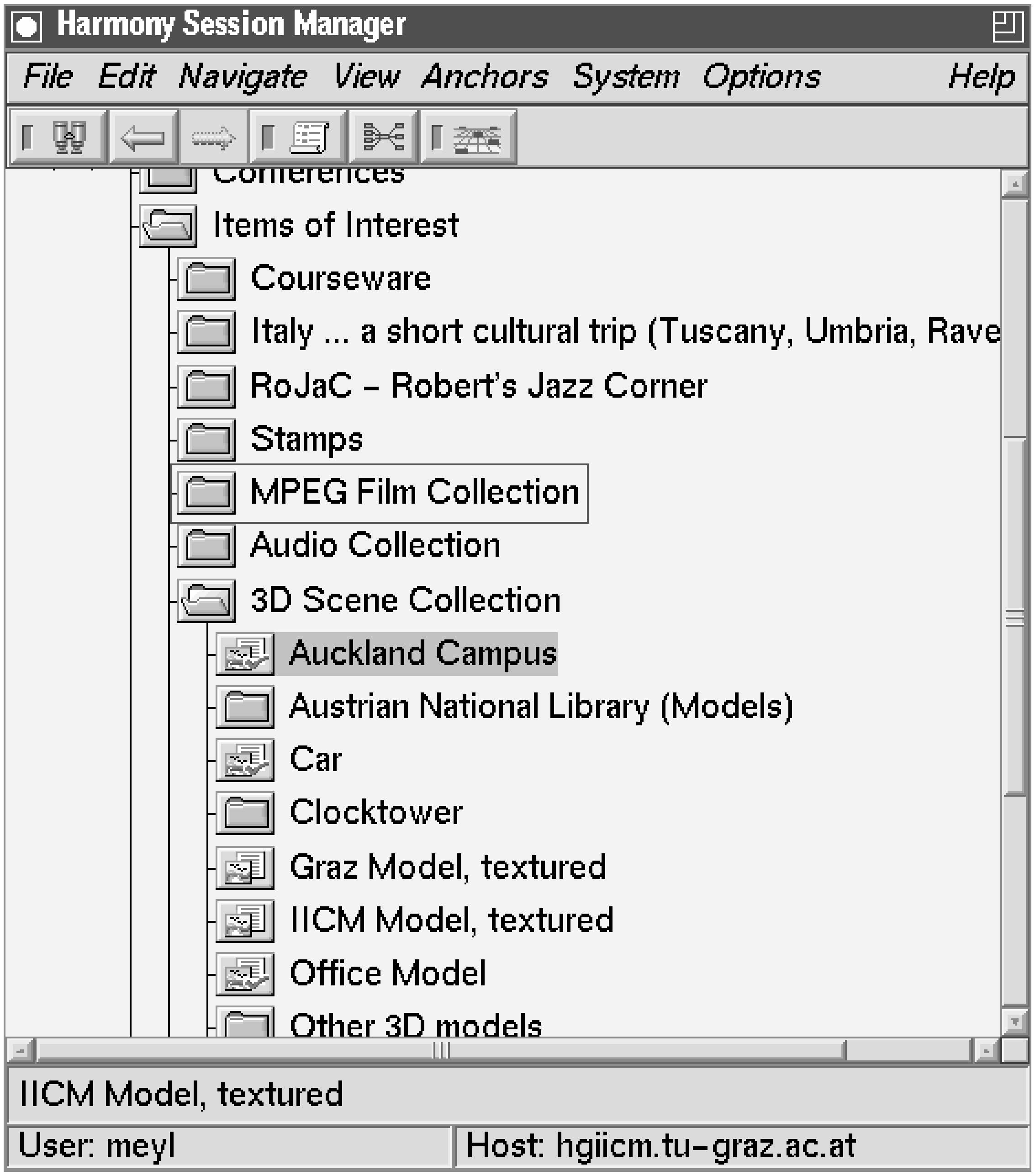
Figure 3.3: Harmony Architecture

3.3 Harmony

Harmony [1] is the native Hyper-G client for X Windows on UNIX platforms. It is written in C++ [30] and InterViews X11 user interface toolkit [17]. Harmony makes the facilities and advantages of Hyper-G available to the Hyper-G user. The collection hierarchy with data from the local and remote servers are arranged clearly and several navigation aids are available. Harmony is an excellent tool to manage large amounts of data and to browse in the information web. Figure 3.3 shows the architecture of Harmony.

Session Manager
The session manager is the primary process. It is connected to the link server and coordinates all activities. The Information Landscape and other navigation facilities are part of the session manager. Figure 3.4 shows the session manager containing a menu bar, buttons, a status-line and a view of the collection hierarchy. The types of the objects (collection, cluster, or document) are represented by a particular icon. The user can open and close collection and view documents by clicking at the titles of the objects. One can also use keys for browsing the hierarchy. Documents which have been visited before in this session are marked with a tick.

The session manager provides several dialog boxes for example a box for identifying or for selecting different settings. In the language dialog box, users can choose their preferred language. The menus, the titles of the buttons, or each message are displayed

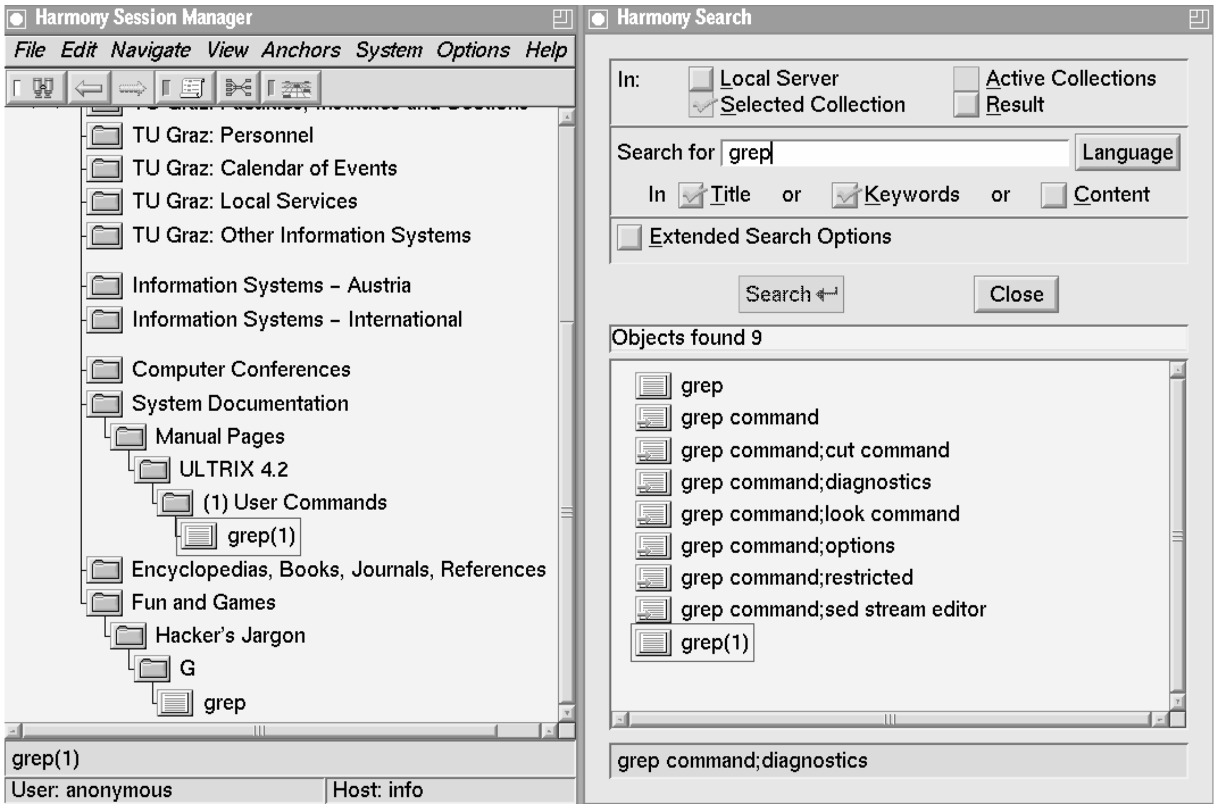


IICM Model, textured

User: meyl

Host: hgiicm.tu-graz.ac.at

Figure 3.4: The Session Manager



TU Graz: Personnel
TU Graz: Calendar of Events
TU Graz: Local Services
TU Graz: Other Information Systems
Information Systems – Austria
Information Systems – International
Computer Conferences
System Documentation
Manual Pages
ULTRIX 4.2
(1) User Commands
grep(1)
Encyclopedias, Books, Journals, References
Fun and Games
Hacker's Jargon
G
grep

grep(1)
User: anonymous
Host: info

Search for grep

Objects found 9

grep
grep command
grep command;cut command
grep command;diagnostics
grep command;look command
grep command;options
grep command;restricted
grep command;sed stream editor
grep(1)

grep command;diagnostics

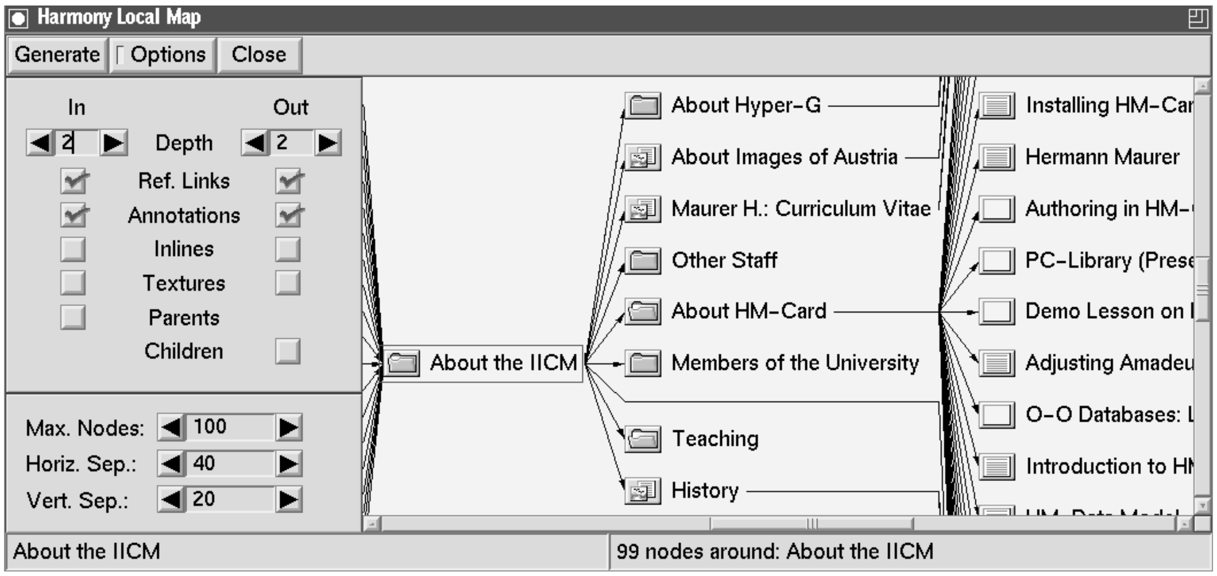
Figure 3.5: The Search Dialog Box

in this language and when the user views a cluster, Harmony chooses the most suitable document from the multilingual documents in this cluster.

One can query the Hyper-G database by using the search dialog box and the dialog box presents the search results in an ordered list. A single click on an object in the list opens the path in the collection hierarchy in order to show the position in the hierarchy. Hence the user can decide whether the document is interesting or not, before actually retrieving it (see Figure 3.5). For example a user who wants to use the command “grep” would choose the document with the topic “grep” in the collection “Manual Pages” and not in the collection “Hacker's Jargon”. The documents can be retrieved and displayed by double clicking on the item in the list. The history list is also active and the user can view a document a second time by selecting it from the list.

Another important navigation aid is the local map which displays documents and hyperlinks (see Figure 3.6). The whole web of links and documents can not be displayed on a single screen. And so the user determines a document and the session manager draws dynamically a desired number of levels of incoming and outcoming links. Documents which refer to the selected document and documents which are referred to by this document can be viewed by double clicking their icons in the graph. The user can also determine the types of the links (reference, parent, child ...) drawn in the local map. The local map is another advantage of bidirectional links.

When the users activates a document, the session manager starts the corresponding



In Depth Out: 2 / 2
Ref. Links: ✓ / ✓
Annotations: ✓ / ✓
Inlines: ☐ / ☐
Textures: ☐ / ☐
Parents: ☐
Children: ☐
Max. Nodes: 100
Horiz. Sep.: 40
Vert. Sep.: 20
About the IICM
99 nodes around: About the IICM

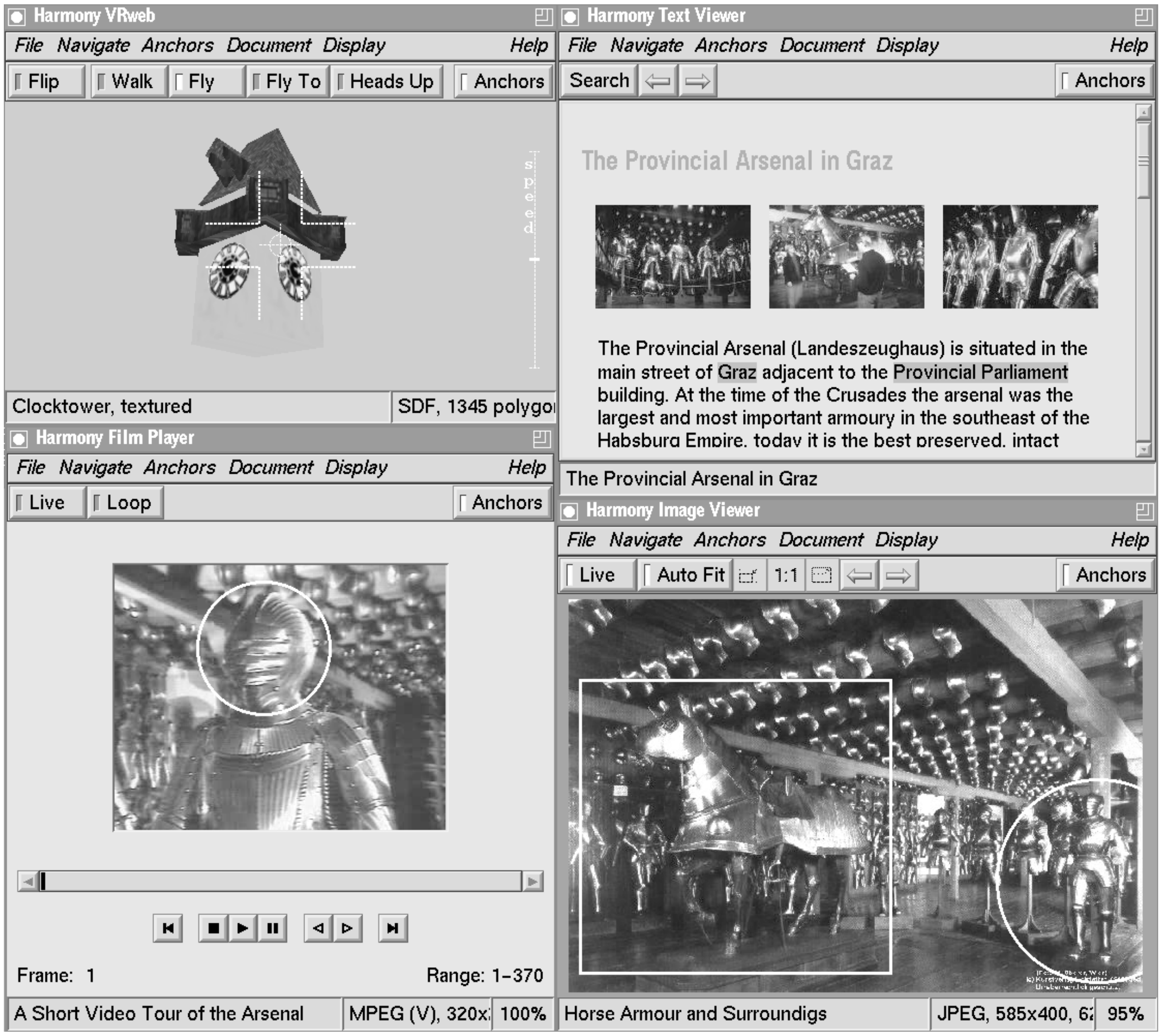
Figure 3.6: The Local Map

viewer for the document. The viewer fetches the document from the Hyper-G server and presents it to the user. Several native document viewers are available in Harmony (see Figure 3.7) :

- text viewer (with inline images)
- image viewer
- film player
- 3D viewer
- audio player
- PostScript viewer

Each of these viewers provide links and one can easily create and delete links. In the text viewer the source anchor of a link is an arbitrarily long highlighted text and in the PostScript viewer links are rectangles in page coordinates. The viewers provide common operations such as scrolling, searching, selecting, etc. One can change the display styles of the various attributes (title, anchor, ...). The image viewer accepts several raster image formats (GIF, JPEG, TIFF) and it has the usual facilities for zooming and panning. In the image viewer, source anchors may be circles, ellipses, rectangles, or polygons. The user can see the source anchor as a highlighted frame. Anchors in the film player are similar to anchors in the image viewer but can move and change size during play track. Source anchors in the audio player are a time period represented by a coloured region at the scrollbar.

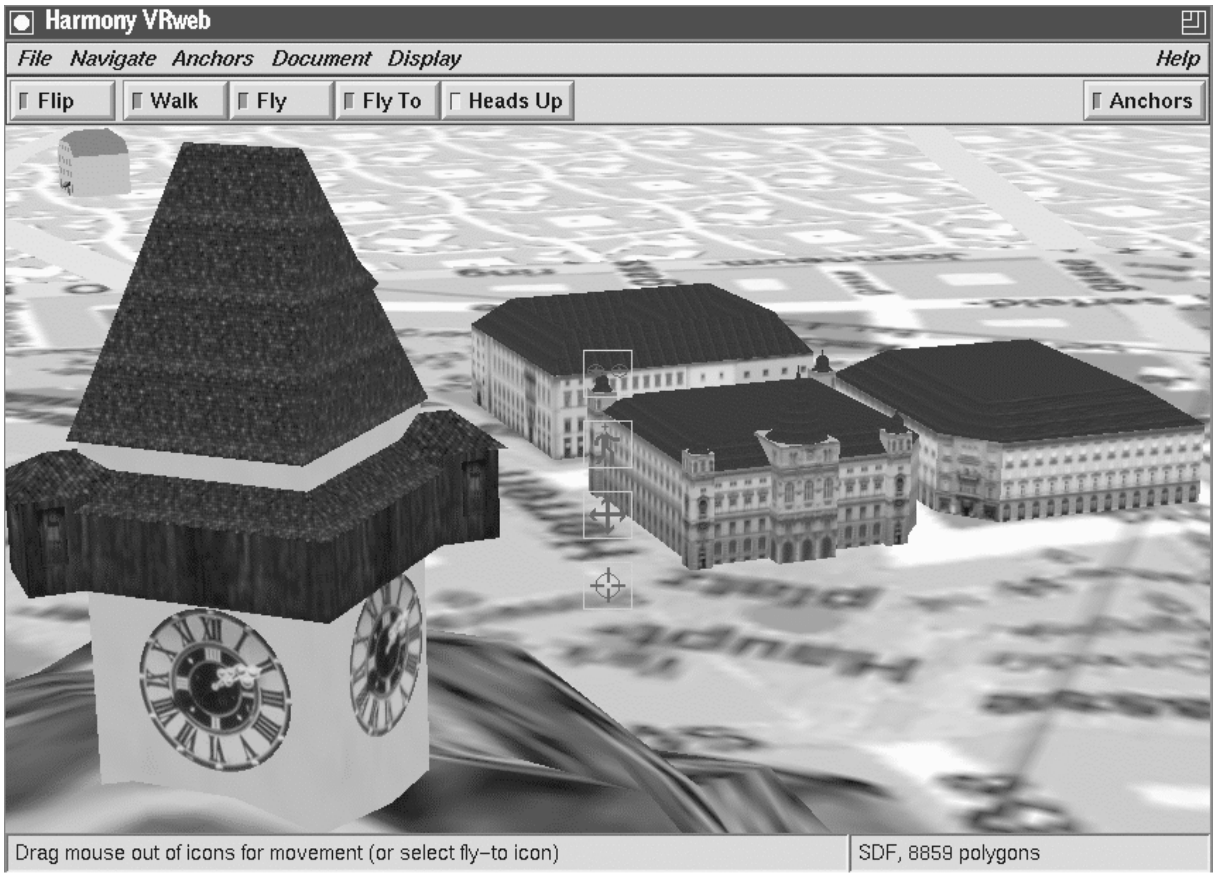
Hyper-G can also be used to store 3-dimensional objects and the native 3D viewer VRweb can display these objects [24, 25] (see Figure 3.8). The user can interactively fly


Harmony VRweb
File Navigate Anchors Document Display Help
Flip Walk Fly Fly To Heads Up Anchors
speed
Clocktower, textured
SDF, 1345 polygo
Harmony Film Player
File Navigate Anchors Document Display Help
Live Loop Anchors
Frame: 1
Range: 1-370
A Short Video Tour of the Arsenal
MPEG (V), 320x
100%
Harmony Text Viewer
File Navigate Anchors Document Display Help
Search
Anchors
The Provincial Arsenal in Graz
Harmony Image Viewer
File Navigate Anchors Document Display Help
Live Auto Fit 1:1 Anchors
Horse Armour and Surroundigs
JPEG, 585x400, 6
95%

The Provincial Arsenal in Graz

The Provincial Arsenal (Landeszeughaus) is situated in the main street of Graz adjacent to the Provincial Parliament building. At the time of the Crusades the arsenal was the largest and most important armoury in the southeast of the Habsburg Empire, today it is the best preserved, intact

Figure 3.7: Harmony Viewers



Drag mouse out of icons for movement (or select fly–to icon)

SDF, 8859 polygons

Figure 3.8: The 3D Viewer VRweb

around (or walk around) in order to explore the objects. There are a lot of possible application for example a complex technical object or a building. Several different navigation techniques are available: object movements and viewpoint movements. In a 3D scene, individual objects can be a source anchor of a link. The 3D viewer provides different methods for highlighting anchor objects. For example colour code (source anchors have one colour, non-anchors another) or bounding cube (anchor objects are enclosed in their bounding wireframe cube). One can follow a link to its destination document by double clicking on an anchor object.

Chapter 4

Spatial Metaphors in Information Systems

The amounts of data which a computer can process and store increase continuously and so human is confronted with more and more information and data. The computer could present these data and information in a very simple form for example lists or sequential printed text and we would have many problems to manage the data or to find a particular part of information. But one can also use increasing computer power to build more comfortable interfaces between computer and human being. Of course, 3-dimensional representation of information needs a lot of computer power but it makes accessing, managing and searching for information easier and one can concentrate upon the creative part of one's work. These are some advantages of spatial metaphors and 3D presentations:

- The advantage of a complex network displayed in 3D space is a simplifying the network structure by removing intersection links. A 3D presentation can reveal the organization or structure of information or a database.
- One can increase the density of information displayed on a single screen by using 3D perspective and so 3D can be used to maximize effective use of screen space.
- User's existing knowledge and natural abilities can be used by using a 3D metaphor. All people are very familiar with navigation in 3-dimensional space and with objects. Users instinctively know what they have to do in a everyday spatial environments and they can draw on existing world knowledge to act in the electronic domain.
- A 3D continuous animation can show complex processes and one can examine every detail of 3D objects by changing the viewpoint.

In this chapter some projects using spatial metaphors are introduced and the chapter provides an overview of the current research work. Several ideas and conceptions have been incorporated into the Harmony Landscape.

4.1 SemNet

SemNet [11] was developed to present large and arbitrary knowledge bases as directed graphs in a three-dimensional space. The elements of a knowledge base are represented as labeled rectangles connected by arcs, which represent relationships between the elements. The user can fly through the three-dimensional space and so can explore the knowledge base. The position of the elements is very important because the user should recognize the organization or structure of the knowledge base. One possibility is to map the properties of an element to a position in three-dimensional space. Another possibility is to place related knowledge elements close together and unrelated elements far apart. Therefore, three techniques were explored: Multidimensional scaling, a centroid heuristic and an annealing heuristic. A further possibility is that the user decides the position of an element. In very large knowledge bases it is not possible to display all elements at the same time. So a form of fisheye view is necessary. A fisheye view means that details are apparent near the viewpoint and only more important context is displayed further away. Using perspective a fisheye view is obtained automatically: A few objects near the focal point, which are points of interest, are very large and those further away appear smaller and smaller. Another possibility is to join elements of less interest (far away) to clusters. The clusters are represented as large rectangles with a different colour. For navigation the user has to know where the current viewpoint is in a complex knowledge space. Therefore SemNet provides overview maps that show the current position of the viewpoint in the x-y and x-z planes. Several techniques for navigation have been explored: relative and absolute movement, teleportation and hyperspace movement. Using teleportation the user can select already visited knowledge element from a list and immediately move to the location of the knowledge element. Hyperspace movement allows the user to follow an arc to a related element.

4.2 Multimedia Information System with 3D Objects

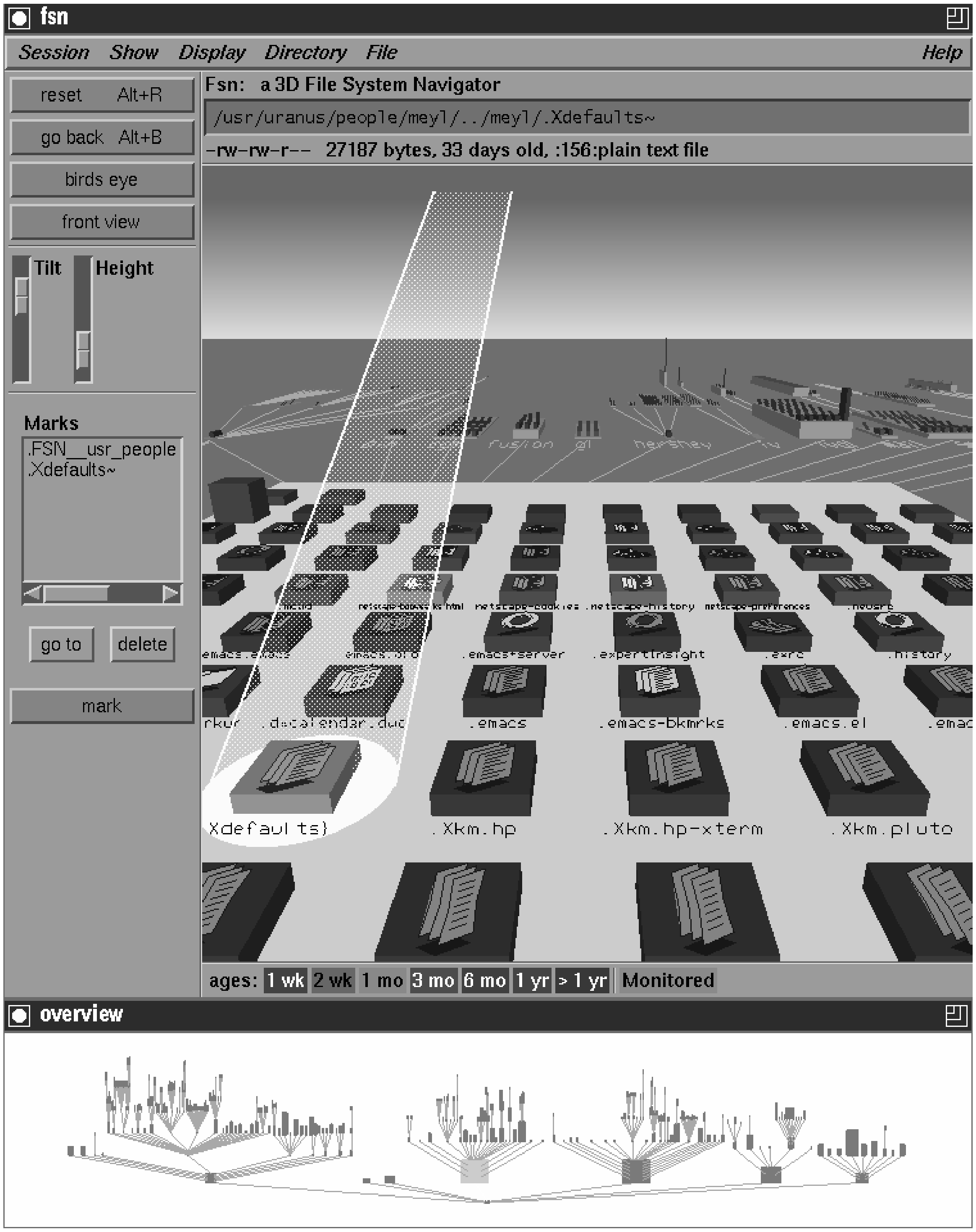
L. Serra et al[27] developed a model for organizing and presenting multimedia information especially suitable to provide information about 3D objects for example an engine or an airplane. An object is a composition of elements and so an object represents a hierarchical structure. For example an engine, a car body, a wheel are part of a car. A piston, a rod, a base are part of an engine and so on. This structure or a 3-dimensional model of the object respectively is the base of the information system. A concept node contains a 3D model of an element, text nodes, image nodes and video nodes. The concept nodes are connected by hierarchical links. Nodes and links build up a tree. The root node contains the 3D model of the whole object and the user can move along the 3D hierarchy to get more detailed information by selecting a part of the object. Association links generates a network of arbitrary text, image and video nodes. Image and video media don't provide a natural procedure of directly manipulating and interacting with their contents. Therefore the existing 3D model is superimposed on the video sequence. The 3D model has to be synchronized in time and space. Now the user can retrieve further information by clicking on the object in the video sequence. The system provides also realistic animations for training applications on mechanical structures. Joint relations are used to describe the relative movement of the 3D objects. And so an animation can be made by using the already existing 3D objects and defining joint relations.

4.3 File System Navigator

The File System Navigator (FSN or Fusion) [31] provides a 3-dimensional graphical representation of the tree structure of a UNIX file system. The tree with the directories and files is stored in a special file which is updated during each session. The directories and files are displayed as blocks. The smaller file-blocks are placed at their directory-blocks which are laid out in an open landscape (see Figure 4.1). The size of the files and directories is mapped to the height of the blocks. A icon at the top of the file-block shows the type of the file and the colour symbolizes the age of the file. Lines, which are drawn on the plane connect directories with subdirectories. The user can fly over the landscape using the mouse. One can also fly to stored places automatically. The user names this places and stores them in the list. Clicking on a line to a subdirectory puts the user to that subdirectory with a smooth animation. A block is highlighted by pointing at it with the mouse cursor. Clicking on the file-block selects that file. The user can recognize a selected block by a virtual spotlight. Double-clicking opens the file. The user can also open another window, which displays an overview map. The user's position in the main window is shown as a red cross in the map.

4.4 Information Visualizer

Card et al at Xerox PARC[26] developed an information visualizer which is suitable for large amounts of hierarchical and linear information. The information visualizer is not only a simple information retrieval system because there are more requirements. The user wants to work (think and act) with the information. Examples are design or decision making. Therefore the time cost of information access must be reduced and the scale of information that a user can handle at one time must be increased. The system provides some methods to achieve these requirements. The heart of the architecture is a controlled resource scheduler (Cognitive Coprocessor) which is responsible for immediate response and continuous animation. For example if the system didn't answer to a search query in about 10 seconds, then the user would get bored and a user can begin the next request as soon as sufficient information has arrived. The system should react to an action (for example a pressed key) of the user in about one second. A room system was implemented because a computer display provides only a limited workspace. Every room is a working space and the user can shift between the rooms. One can view an overview map which shows all rooms at the same time. Card et al implemented also 3D representation of information and animation to increase the density of information displayed on a single screen or in a room and so 3D can be used to maximize effective use of screen space. The cone tree presents hierarchical information structure. The information chunks which are called nodes are displayed as a index card with a title in 3D space. This nodes build up a tree. Each layer of nodes in the tree is drawn below the previous layer, with their children in cones. The index cards of the nodes are placed on a cylinder at the button of the cone, which can be rotated by selecting a node. The rotation is animated so the user sees the transformation. The developers used the cone tree to display a UNIX directory hierarchy, which contained about 600 directories and 10000 files. A 2D tree of this hierarchy could never displayed on a single screen. The perspective wall presents



Fsn: a 3D File System Navigator

/usr/uranus/people/meyl/../meyl/.Xdefaults~

-rw-rw-r-- 27187 bytes, 33 days old, :156:plain text file

.FSN__usr_people
.Xdefaults~

Figure 4.1: The File System Navigator

linear information structure with spanning properties (for instance files sorted by time). The information is presented at a wall which uses the fisheye view. The wall slides the item of interest to the center panel with a smooth animation where the item is displayed with great detail. The perspective view makes the neighborhood of the "area of interest" larger than more distant parts of the contextual view.

4.5 Navigational Aids in Hypertext

Smith and Wilson [29] examined the problem of navigation in hypertext and developed four types of navigation aids. Hypertext systems are a kind of database in which related documents are linked together. That means a lot of advantages over other databases, but hypertext may also present a new set of problems. The major problem is the disorientation or becoming "lost in hyperspace". Another problem is for example to digress from the main task because of interesting information on the way. Smith and Wilson inspected different types of navigation or orientation aids like indices, typographical cues or graphical browsers. These aids are insufficient for large hypertext systems. Therefore Smith and Wilson described a hypertext system which can represent spatial and schematic forms of the network in two or three dimension, using HyperCard[1] and Virtus Walkthrough[2] . In a schematic network the position and distance of the nodes and links have no meaning, however in spatial networks nodes with related content are close together. Smith and Wilson chose for the test domain an academic department information system. A conventional 2D browser with a schematic representation of the network and a spatial representation in a form of a map of the department are available. The user's position in the network is highlighted. A 3-dimensional schematic representation of the network removes the intersection of the links and is very useful for large networks. The 3-dimensional spatial representation is a 3D model of the department and the user can walk through the rooms of the building. For example in the offices the user can find information about the members of staff and their research interests. In the laboratory are information about the facilities and so on. Another possible application may be manufacturing shop-floor information system or a public building direction information.

4.6 Bead by Rank Xerox EuroPARC

Bead [9] is a prototype system for graphically-based information retrieval, where sets of documents are stored and categorized in order to allow for search and retrieval. Documents are close in 3-dimensional space if they have roughly the same words occuring in the 'keywords', 'title' and abstract sections, and so spatial distance corresponds to thematic similarity. In an earlier version of Bead this information was represented as 'point clouds' in a 3-dimensional space, but users found it difficult to get an overview of the entire set of documents. It was also difficult to orientate themselves and navigate within the space. And so an open landscape was developed, where the documents are represented as objects on a ground plane. The user has an overview of similarity and dissimilarity of the docu-

[1]HyperCard are trademarks of Apple Computer Inc.

[2]Virtus Walkthrough is a computer-aided visualization system (Virtus Corporation, N.Carolina, USA).

ments making up the corpus. Clumps of objects, gaps and a surrounding contour serve as natural reference points (or landmarks) and are important in orientation and navigation. Initially a user makes a search for a keyword and the matching documents change their colour. The resulting patterns of colour show the distribution of matching documents in the corpus of documents. Then maybe the user finds other documents nearby, which are more interesting, and he starts a search with a different keyword. Initially known documents may be dispersed among other unknown but potentially relevant ones.

4.7 LyberWorld

LyberWorld [18] is a 3-dimensional graphical user interface for an information retrieval system. Matthias Hemmje used a database with 800 scientific publications. This system is able to find automatically documents which are relevant for a particular term. A user can start the search with a word and then he can examine the relevant documents and start another search with a different word and so on. A new user interface has been developed for this searching process or search-path. Hemmje represents this search-path as a hierarchical 3-dimensional tree called a cone tree. A document is specified by several terms which are the children of the document. A term has some documents as children which are relevant for this term. The root of this tree can either be a familiar document or a term (the start of the search) and then there are alternate term-levels and document-levels. A document or term with its children is displayed as a cone. The user is allowed to open or close subtrees as he likes and so he examines only the interesting part of the document corpus. The titles of the documents and the terms are placed on an cylinder, which can be rotated. The user can enter a room by zooming into a document-symbol and the whole document is projected on a wall. And so he or she can decide whether the document is relevant.

4.8 Information Island

Increasingly we are confronted with information in the electronic world of distributed and networked computers. Therefore J. Waterworth and G. Singh [32] developed a user interaction model to be able to navigate and orientate in this complex, electronic world-at-large. In this model, the world-at-large is represented as a set of archipelagoes, each is a collection of information island. An archipelago represents a major class of service or application, for example information services, communications, medical and financial services. Each island generally contains only one subclass of information. On an island are buildings and in each building are a set of information sources or services related to a particular theme. Examples might be Weather Building, Sports Building, Stocks and Shares Building. In all buildings are common features which allows the user to get an overview what is available in a building. Now the user can explore this world with a vehicle, that provides a private and a public view. The public view or the "God's Eye" view shows everything that is available. The private view contains only for the user interesting objects. Both views can be displayed simultaneously in two windows on the screen and so the user can easily select an object (for example an island or a building) from the public view for his private view. It is also possible to store and revisit without navigation a list of places. There are different types of agents. The user can order information about what is available and where it is located from an agent. The system provides three kinds of navigation: moving around the world and in history, viewing different levels of the world. For example the user may want to see the whole world and then examine an island of interest and so on.

4.9 Mapping from Spatial Cognition to Hypertext

The navigation in a hypertext is similar to find ones way within real world environments. And so Shum [28] inspected the results of spatial cognition research and he simulated an interface for hypertext using this results. The main problem in hypertext is to maintain the orientation (not to get lost) and therefore the user has to acquire and develop a cognitive map. A cognitive map means an orienting schema or a mental representation and one develops it by browsing the information web and/or with a graphical map of the network. There are four different classes of information which embodies a cognitive map. Locational information are distance and direction and attributional information are descriptive and evaluative attributes. Descriptive attributes tell the user something about the content of the node. With the evaluative attributes of a node one can decide whether a visit is important for him or her or not. Inhabitants sketched maps of cities (which reveal some of the content of cognitive maps) consist of landmarks, paths, nodes, districts and boundaries. However, the last two terms are often underestimated in hypertext. If the spatial environment becomes dynamic (for example different views) it is very difficult to acquire a cognitive map. It is also important for developing a cognitive map that a node's location carries information about the content. Shum developed a layered-space model for certain tasks (no model is suitable for all applications) by considering the preceding analysis. The 3D euclidian space is structured into layers and subspaces between layers. Each layer contain a root node of a subnetwork which is dedicated to a particular subtopic of the complete information space. The subnetwork is located on the layer or in the associated subspace. These nodes are subspace nodes. One can click on a subspace node and a new window shows another subnetwork of nodes. Thus there are three levels: layered representations, subnetworks, and actual nodes. If a subnetwork in its own window has links to other parts of the hypertext, the location of the remote nodes is represented by a schematic icon. The property of a layer or a node can be mapped to its position, for example the further to the right layers are, the more advanced the material becomes.

4.10 Web World

Web World [7] uses the metaphor of a landscape with houses and presents a part of the WWW hypermedia web. Each user can have his or her own house and clicking on this house activates the link to the user's WWW server. This town doesn't show the whole current web because the set up of a house is voluntary. One can only move forward, backward, left and right and one always looks from above on the landscape. And so interactive movement for example changing the line of sight or the height is not possible.

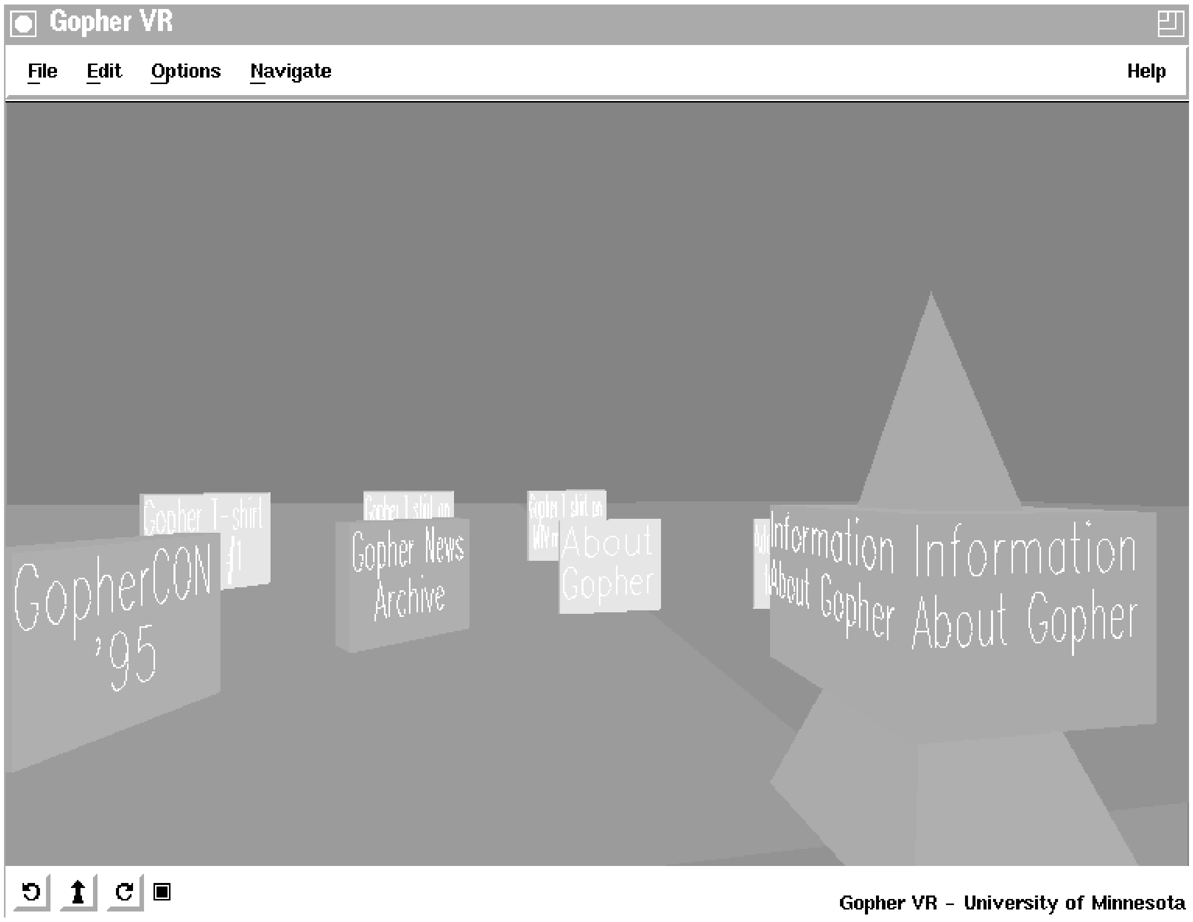


Figure 4.2: GopherVR

4.11 GopherVR

Gopher is an Internet-based information system and provides a hierarchical structure (menus) to access data which are stored on different servers (see also Section 2.4.3). The current interface has a lot of limitations (the menu-items are presented in a linear list) and there are three usage problems: lost-in-space problem, the grouping problem and the browsing problem. The users often feel lost after browsing for a while and they don't know where they can find the desired information. The grouping problem means that it is difficult to show relationships between menu-items in a linear list and the browsing problem occurs because the title of the menu-item reveals only little of the contents. Hence McCahill and Erickson [20] developed a 3-dimensional interface which provides a more comfortable and natural way to browse the hierarchical menu-structure. The items of a menu are arranged circularly which means a number of advantages (see Figure 4.2). For example, users generally have a head on view of the title on the fronts of several 3D icons and if one is inside the circle then one always looks at something. A 3D 'kiosk' icon at the center point represents the parent menu and provides a link back to the previous menu. The result of a search query is presented in a spiral which provides a natural ordering for the relevance of the items. A menu-item is represented by a 3D icon and the basic form is an approximately rectangular box. The form and the colour shows the type of the object (sub-menu, document, interactive session and so on). The title of the item and a proxy is displayed on the icon and the proxy reflexes something of the contents. Overview maps should give an overview of a menu and an overview of the local region of Gopherspace.

The program provides several navigation modes and it supports special movements like circular motion. Of course the new client is backward compatible and can access the data of any Gopher server. Newer servers provide additional information about the menu-items.

Chapter 5

The Harmony Information Landscape

The Harmony Information Landscape is one of several navigation aid in the Harmony client for Hyper-G and it is used to maintain orientation in the large information universe. It supports a user interface which is very comfortable and easy to handle.

The Landscape is part of the collection browser and provides another view of the current collection tree but it does not portray links. The spatial metaphor used in the Landscape is an open landscape where objects (collections, clusters and documents) are spread out. User's existing knowledge and natural abilities can be used by using such a 3D metaphor. One can interactively fly over the landscape, open and close collections and view clusters and documents.

5.1 Layout

We live in a world of three physical dimensions and we are used to move in this world and to perceive individual objects. But the horizontal extent appears larger then the vertical because we are on the surface of the earth. Chalmers [9] calls our world '2.1-dimensional'. And so we are very familiar with an open landscape wherein different objects are placed. The third dimension 'height' can be used to encode an attribute of the object. Some projects already used this metaphor like Bead [9] or the File System Navigator [31].

The Harmony Information Landscape (Figure 5.1) presents the collection hierarchy of Hyper-G in an open landscape. There are no boundaries in this landscape and the collection tree can be arbitrarily large (practical depending on the computer power). It can spread out over an unlimited area. A horizon and a sky give the realistic impression that the collection tree is placed on the surface of the earth. The sky is either blue or sunset glow (continuous from blue to red) with some stars. The colour of the sky is adjustable by X attributes. The sunset glow on the horizon has the appearance of a landscape in the evening. The collection hierarchy itself is an acyclic directed graph (not a tree) because a document or collection can have more than one parent collection. But the hierarchy which is shown in the landscape is really a tree. A document or a collection with two (or more) parent collections is displayed two (or more) times. Each parent collection has its own child and the same object can be found more than once in the tree.

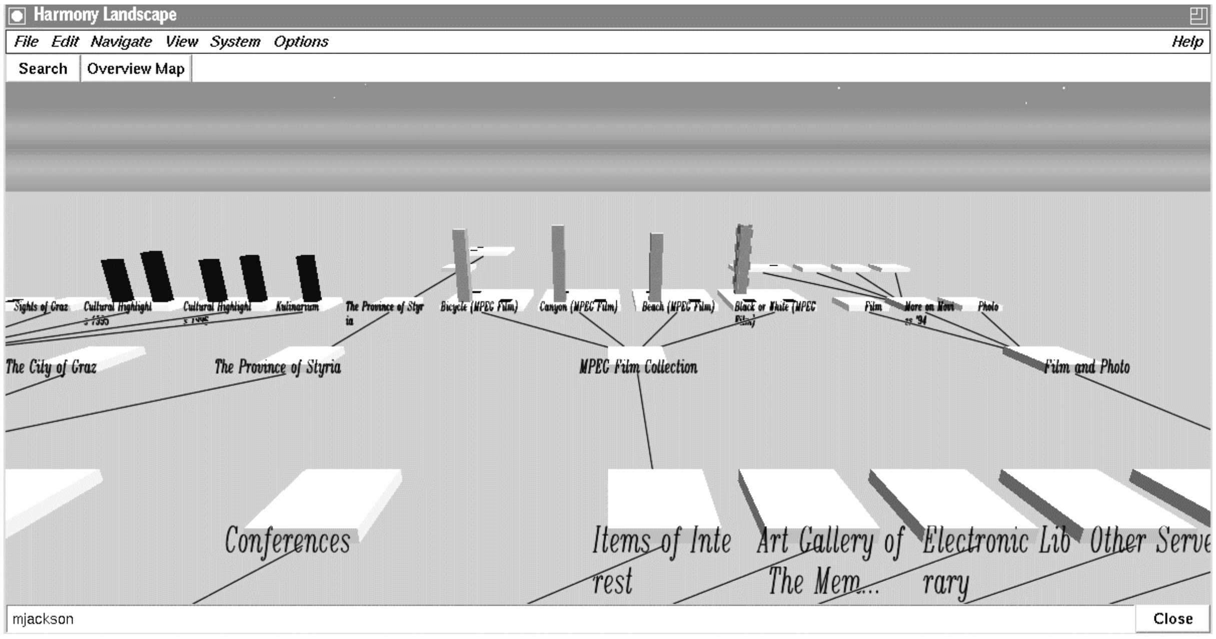


Figure 5.1: The Harmony Information Landscape

A document is represented by a block. The size (n bytes) of the document is mapped to the height of the block. The size of a typical text document is very much smaller than the size of a long MPEG movie. Hence a logarithmical function is used to calculate the height of a document block.

A document is always part of a collection or a cluster which is also displayed as a square block. This block is larger than the document block but it is less high like a socle. The document blocks are placed on their parent collection or parent cluster. And so the user can easily comprehend the contents of a collection. The subcollections (the collections in a collection) are symmetrically located behind the parent collection one level further. They are connected by edges to the parent collection. The subcollections carry their own document blocks. The collection socles with their documents on the top are spread out in the landscape. The type of an object (collection, text, image etc.) is mapped to a colour which is adjustable per X attribute. For example, the user can determine that a movie document should be red and a Postscript document blue and so on. Now the users can know the type of the document but they aren't informed about the contents. Therefore each block is inscribed with a title. The title can be very long and for that reason it extends over two lines. If the space is too small for the whole title ellipses (...) are appended to indicate that the title has been truncated.

Texturing means that an arbitrary 2D bitmap is mapped onto a surface in 3D space for example on the plane, the horizon or on the surfaces of an object and it can be used to enhance visual realism of a 3-dimensional world. At present the user can determine four different textures per X attributes for the plane, the background, collections and clusters. A marble texture is used for collections (per default) so that a collection has the appearance of a marble socle (see Figure 5.2). The pattern of a stone bottom which

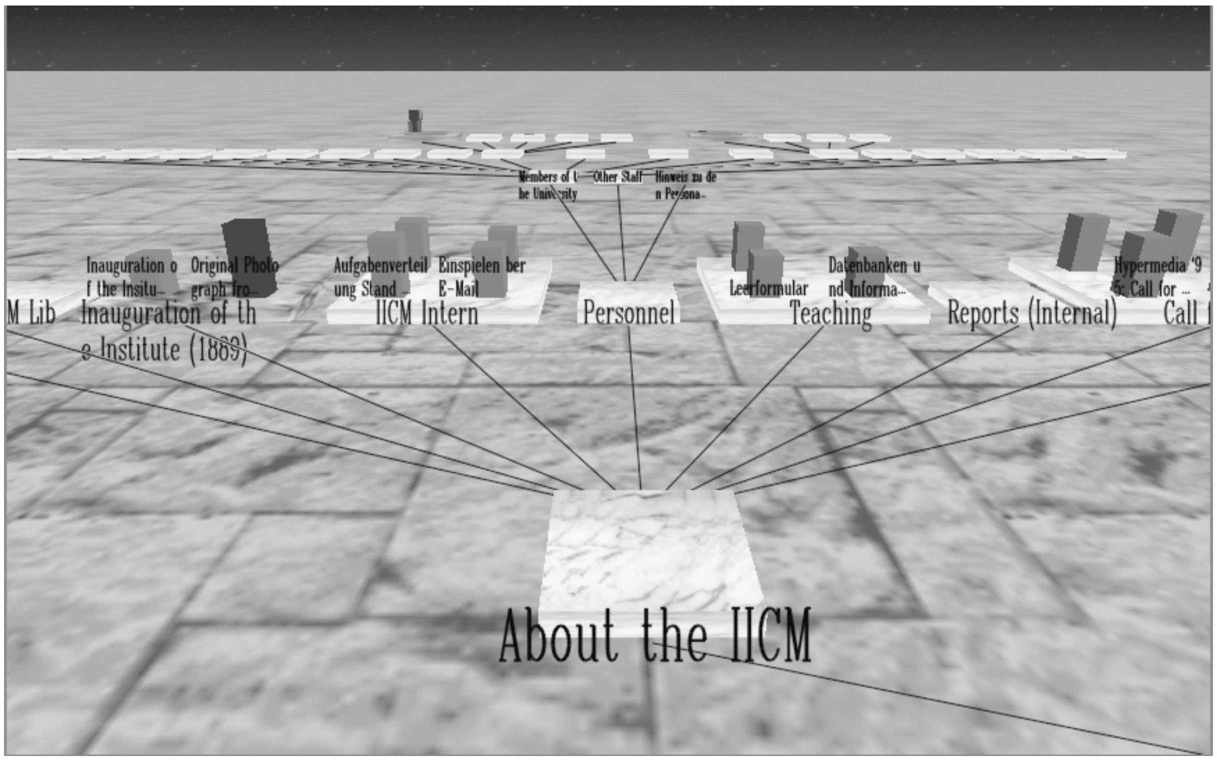

Members of U he University
Other Staff
Hinweis zu de n Persona...
Inauguration o f the Insitu...
Original Photo graph fro...
Aufgabenverteil ung Stand ...
Einspielen ber E-Mail
Leerformular
Datenbanken u nd Informa...
Hypermedia '9 5: Call for ...
M Lib
Inauguration of th e Institute (1869)
IICM Intern
Personnel
Teaching
Reports (Internal)
Call
About the IICM

Figure 5.2: Textured Landscape

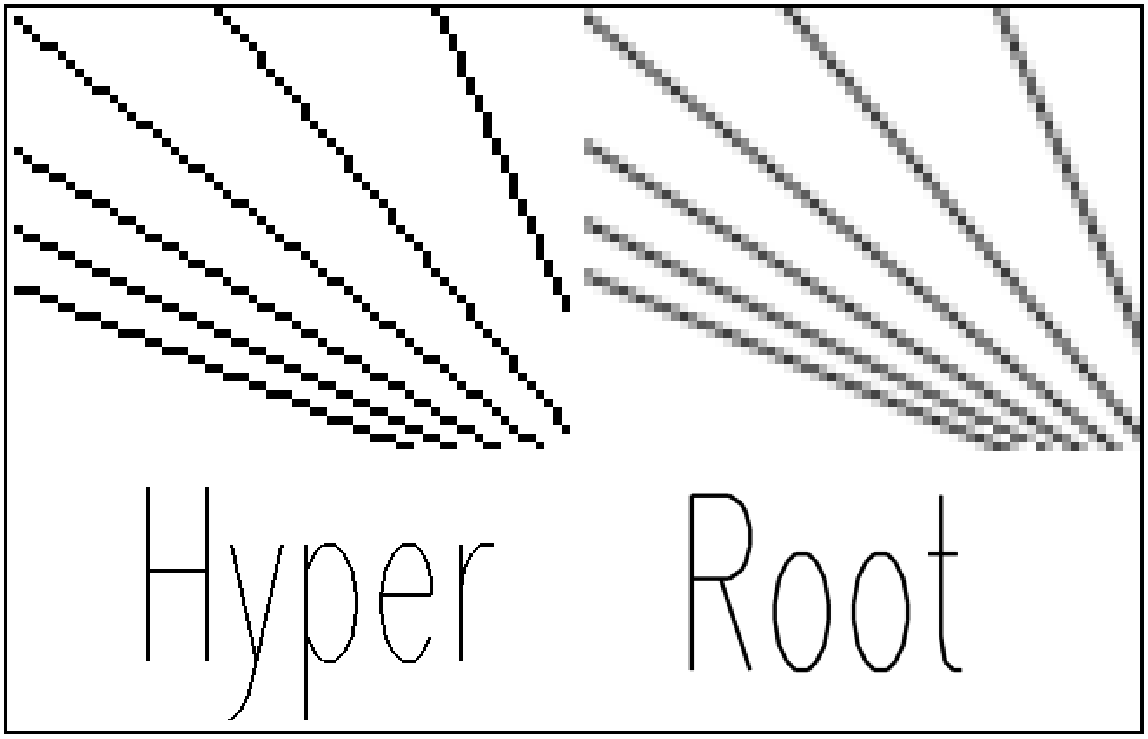


Figure 5.3: Lines and Letters without (left) and with (right) Line Antialiasing

is used for the plane has another advantage: The movement of objects in the landscape indicates that one is moving. One doesn't become aware of a movement when there are no objects. In the landscape with a pattern on the plane one always becomes aware of a movement and the moving stone bottom makes the impression that one is flying.

Line antialiasing is a method that makes lines drawn on a discrete device (the display screen with discrete pixels) appear smooth and it is used in the Landscape to improve the readability of the lettering. Figure 5.3 shows letters and an enlargement of lines with and without line antialiasing. Line antialiasing and texturing can only be used with powerful graphic computers.

5.2 Hit, Select and Activate

The user can touch each object (even the edges) in the landscape by moving the cursor over the object. When the user touches an object, it is highlighted which means that the edges of the block and the edge to the parent collection is drawn in a particular colour. The status-line shows the title of the highlighted object.

When the cursor is not over an object, no object is highlighted and the status-line shows the title of the current or selected object. The edges of a current block are drawn in a distinct colour and are dotted lines. The user can select an object by clicking the left mouse button on the object and this object remains the current object until the user selects a new object. The Landscape and the collection browser are coupled and so the current objects in the Landscape and in the collection browser are always the same (see Section 5.7).

Single-clicking selects an object and double-clicking activates an object. When the user activates a document, the Landscape starts the corresponding viewer which presents the document. Activating a cluster shows all documents which belongs to the cluster and so a cluster is a kind of multimedia document. Double-clicking on a closed collection (no subcollections are visible) opens the collection and shows all subcollections at the next level. When the user double-clicks on an open collection, all subcollections are removed recursively (the whole sub-tree). For example after closing the root collection only one block (the root) is visible in the landscape.

If one opens or closes a collection, the layout of the tree will be changed. The positions of the collections and clusters have to been calculated newly and the positions can differ from the old positions. The Landscape presents a new tree. That can mean that a collection or a cluster abruptly disappears in the window and then the user may be confused. Therefore the Landscape doesn't change the position of the current (open or closed) collection or cluster[1], because the users focus their attention just on this object. Other objects may change their position but the users can see always the current object at the same position after calculating a new tree.

Another solution would be to present the whole tree with all collections, clusters and documents, for which the Landscape has to read the whole collection hierarchy from the server, store the data local and update the data regularly. But there are several problems with large amount of data for example speed problem depending on the computer power (see also Section 6.5). And so in this version the users can open and close subtrees as they likes. When the users search for a particular document they successively open collections with analogous topics until the desired document is found.

5.3 Overview Map

The Landscape provides an overview map which presents a general view of the current collection tree from above (see Figure 5.4). When the user selects the menu item overview map, the Landscape opens a new window with the map. The window can be placed as the user likes and it shows all collections and clusters but no documents and lettering. When the user hits an object with the mouse cursor, the status line shows the title of the object and the user can also open and close collections in the overview window. The user's position in the main window is represented as a red cross in the overview map. The larger the tree, the smaller the tree is shown in the window with unchanged dimensions. If the tree becomes too large, only the area surrounding the user's position is shown in the overview window and not the whole tree. In that case a movement in the main window shifts the presented area in the overview window.

Comparable to a road map the users can use the overview map to find their position in the information tree and to find out the direction for reaching a new goal. The user can also perceive the shape and extension of the current collection tree and the overview window provides another kind of navigation (see Section 5.4.3).

[1]Actually the viewpoint is changed.

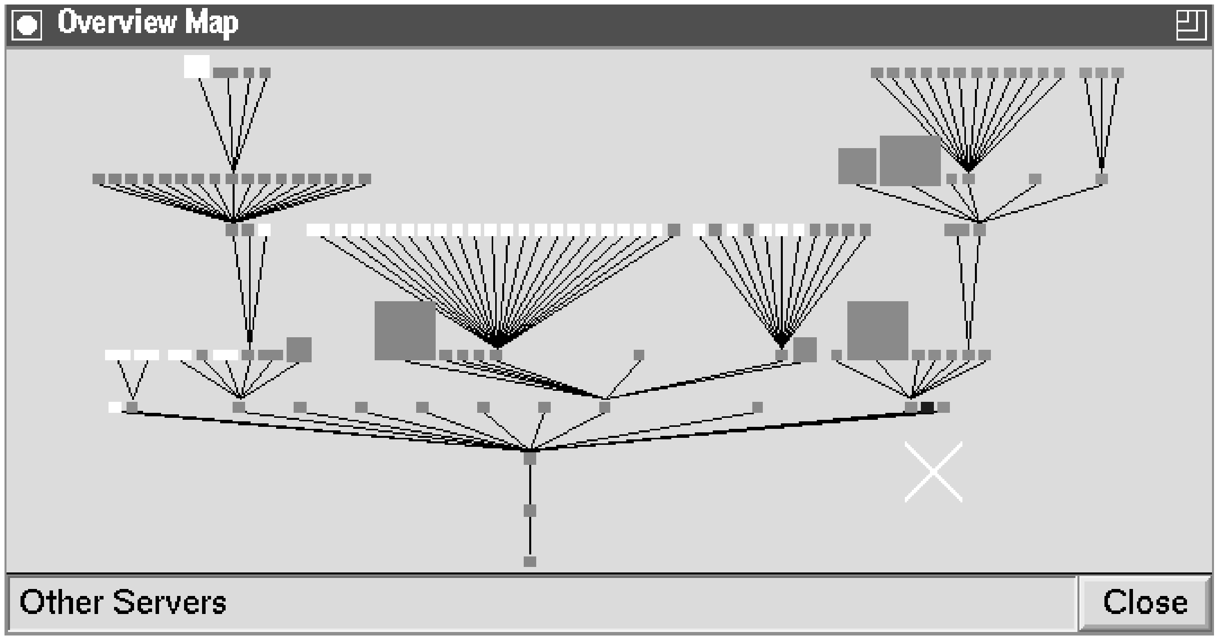


Figure 5.4: Overview Map

5.4 Navigation

The information tree is located on the surface of a 3-dimensional world and the user's desire is to change his or her position in order to explore this world. One distinguishes between two basic methods: moving the objects or moving one's viewpoint[24]. The first method is most used to examine a single object for example a complex technical engine. Moving the viewpoint and changing the orientation is equivalent to the movement in the real world and is used in the Landscape. Users feel as if they are flying through a virtual world. The Landscape provides different types of viewpoint movements: *General movement* can be used for exploring the information tree. *Targeted movement* is used for moving to a particular place or object for example an object with a particular content (searching), or object which has already been visited (history), or an object which the user can see.

The use of the input devices should be easy and easy to learn and the user should be able to work with the program without specific knowledge and training. Natural abilities and knowledge should be used. The 3D input devices (data glove and data helmet) fulfill these requirements. For example everybody instinctively know that one has to turn one's head to change the line of sight. The data helmet can track the orientation of the head and change the orientation of the viewpoint. But these input devices are not widely in use at present. A mouse is used with almost every computer and most users know how to use a mouse. Hence, the Landscape uses the mouse and the keyboard as input devices. But the mouse is only a 2-dimensional input device and therefore a 2-dimensional mouse movement has to be mapped to a 3-dimensional movement. The result of a mouse movement can be changed by using the mouse button or the keyboard and so 3D movement is possible.

In the real world we can change our position (three dimensions) and orientation (three dimensions). In the Landscape users always look straight ahead and can not turn around. One can only move forward, backwards, left, right, up and down and can change the view direction between horizontal and vertical (angular of 90 degrees) which means several advantages and maybe some disadvantages:

- One has to learn and know less movement commands and so a novice is not so confused. Using a program with complete freedom of movement, untrained handling can cause an adverse viewpoint and orientation for example upside-down or looking in the sky. Then the user may not find back to a normal position. That can not happen to a user of the Landscape. Complete freedom is not necessary, because the user is only interested in a particular area (the collection tree) and not for example in the sky in contrast to Harmonys 3D viewer VRweb which supports complete freedom of movement.

- Another advantage is that the objects behind the viewpoint can be determined without complicated calculations. These objects must not be taken into account for picking and drawing. Objects further away can also be determined very easily and drawn with less detail.

- Most importantly one doesn't need the z-buffer. (see also Section 5.8).

A possible disadvantage is that one can not move as one likes for example turn around and look back and the direction of moment is not always the current view direction. When users move backwards, they can not look in the direction of flight.

The Landscape provides four navigation modes:

5.4.1 Fly

Fly is the first and only general movement in the Landscape and the navigation mode is called fly, because one can walk not only on the ground plane but also one can fly up in the sky comparable to a helicopter (The great advantage of a virtual world is that one can do thing without cost, money or specific knowledge). Using the mouse and the buttons two parameter are controlled :

- direction of flight

- speed

The direction of flight is not always the view direction. In the real world we normally move in our view direction but in the landscape the user always looks straight ahead and a movement command always causes the same movement direction independent of the view direction. For example the command 'forward' moves the user forward although he is looking down. Therefore a novice might be not so confused.

All kinds of motion are achieved by dragging the mouse while pressing a button. When a mouse button is pressed, the Landscape draws a cross at the position of the mouse cursor. This cross indicates the reference point. Now the relative position of the cursor to the reference point (the cross) determines the direction of flight and the speed. The speed is proportional to the distance between the reference point and the cursor and the direction from reference point to the cursor determines the direction of flight. Table 5.1 shows the mouse button assignment.

The left mouse button and middle mouse button change the position of the user (viewpoint) and the right mouse button only changes the view direction and not the position. The user can immediately stop the movement by releasing the button.

mouse button	*mouse movement*	*resulting movement*
left	up / down	flying forward / backward
left	left / right	flying left / right
middle	up / down	flying up / down
middle	left / right	flying left / right
right	up / down	tilt the head up / down

Table 5.1: Viewpoint Movement

When the user looses the orientation, he or she can choose the menu item reset to come back to the root collection.

5.4.2 Point of Interest Movement

The point of interest (POI) movement or fly-to movement is a target movement. The user can choose any visible target (point) in the landscape and can move towards this point of interest. The POI movement uses a logarithmic function in order to calculate the speed. The current distance between the viewpoint and the POI determines the speed. This means that the movement is fast towards the POI and slow near the POI. For example, imagine the user wants to read the title of an object, which is very far away. Using POI movement at the beginning the user covers a large distance very fast. The target object appears to grow at a constant rate during approaching the viewpoint. The movement is slow near the object and the user can easily find the desired position in order to read the title. It is impossible to fly through or into the object.

The user can choose the desired point or object with the mouse cursor. He or she starts the POI movement by pressing the middle (right) mouse button and the CTRL button. The Landscape draws a crosshair symbol at the position of the POI. The viewpoint moves towards the POI by pressing the middle mouse button and away from the POI by pressing the right mouse button. One can stop the movement by releasing a button.

5.4.3 Navigation using the Overview Map

The overview map presents a general view of the current collection tree and it provides another kind of navigation (targeted movement). One can select any place in the overview window by moving the mouse cursor to the desired place and pressing the left mouse button. Then the viewpoint in the main window moves towards this place and the cross in the overview map, which indicates the viewpoint position in the main window moves towards the mouse cursor. The line of sight doesn't change. When the user arrives at the desired place, movement stops. One can also move the mouse cursor while the mouse button is pressed. The cross always follows the cursor. So the user can move to any precise position in the landscape.

5.4.4 Animation

Sometimes the user wants to know the position of a particular object in the collection hierarchy or to move to a particular object. The Landscape solves this problem by putting the user to the desired object with a smooth animation. Card et al[26] emphasize the importance of smooth interactive animation: " ... because it can shift a user's task from cognitive to perceptual activity, freeing cognitive processing capacity for application tasks." If the Landscape immediately put the user to another position without animation, the display would jump from a configuration to another. The user would not know the direction he or she has moved and would have to spend time (and maybe cognitive effort) reassimilate the new view and finding out his or her new position in the tree. Using the animation one can see how one's position is changing.

One can select an object and the Landscape leads the user from the current position to the desired object. There are some possibilities to select an object and to start a smooth movement towards this object :

- Single-clicking the left mouse button on an item in the history list or in the search result list (see more in Section 5.4.5).
- Double-clicking the left mouse button on an object (which also activates it).
- Double-clicking the left mouse button on a line on the plane selects the collection or cluster, which is at the end of the line (but doesn't activate it).
- Double-clicking the middle mouse button on an object selects it's parent collection (but doesn't activate it).

The last two points are very useful for moving along the collection tree. One can move forward by selecting the desired subcollections and one can move backward to the parent collection (up to the root) by using the middle mouse button. The Landscape also whisks the user to the current object by pressing the space button on the keyboard.

The fly animation can be immediately stopped by pressing the left or the middle mouse button. Pressing the left mouse button leaves the user at the current position, pressing the middle mouse button puts the user (without animation) to the desired object at once and so the user has not always to wait for finishing the animation. There are two possibilities for the view direction and the height of the viewpoint during the animation. One can change the height and view direction, so that, upon arrival, the user has a good view of the object and can read the title. But maybe the user doesn't want to change the height or the view direction, he or she just wants to move to the object. So the user must always rechange his or her height and view direction, which can be very tiresome.

The alternative is that the height and the view direction is not changed and maybe the title is not readable. The Landscape changes the height and view direction when the user deliberately wants to move to a particular object (the last two items in the list above) and it doesn't change the height and view direction when the user opens or closes a collection or clicks on an item in the history or in the search result list.

5.4.5 Searching and History List

The collection browser provides search facility, backtracking and history list as navigation aids which are very important for the users to find their way in the information net of links and nodes. These navigation aids can also be used to move in the Landscape and are additional navigation modes. Each activated object (document, cluster or collection) and each search is stored in the history list. One can view this list by selecting the menu item or pressing the button 'history list' in the window. The list shows the titles of the objects and an icon which symbolizes the type of the object. One can select an item in the list by pressing the left mouse button. In the Landscape this object becomes the current object and the user flies to this object with a smooth animation. So the history list is a kind of memory for already visited places.

The result of a search (the matching documents or collections) are presented in a list. This list is similar to the history list and one can also select a item by a single-click. In case the selected object should be not part of the current collection tree, the Landscape inserts the object and all parent collections recursively. When the selected object or any parent collection has more than one parent collection the program accidentally chooses one parent collection and inserts it in the collection tree. The Landscape puts the user to the selected object with an animation.

5.5 User Interface

At the top of the Landscape window there is a menu bar with pull-down menus and a row of buttons (see Figure 5.5). At the bottom of the window there is a status line (with a progress indicator) and a button which is used to close the window. When a collection is opened, the progress indicator shows the expected time the user has to wait until all children are retrieved. The menu is very similar to the menu of the collection browser but some menu entries only concern the Landscape. Most of the commands can also be executed by pressing a particular button. The following commands and dialogs which concern the landscape are activated with pull-down menus: reset the view, colour chooser, font chooser, scale dialog, switching on/off texturing and line antialiasing and save the current collection tree in a file using the VRML format (at the moment without lettering). VRML is platform-independent markup language (Virtual Reality Modeling Language) [23, 4]. The standalone version of the Harmony 3D viewer [25] can read this file and one can explore the collection tree with the 3D viewer. But it is not possible to open or close a collection or to view a document.

5.5.1 Font Chooser

Figure 5.6 shows the Font Chooser of the Landscape. One can choose a font by clicking

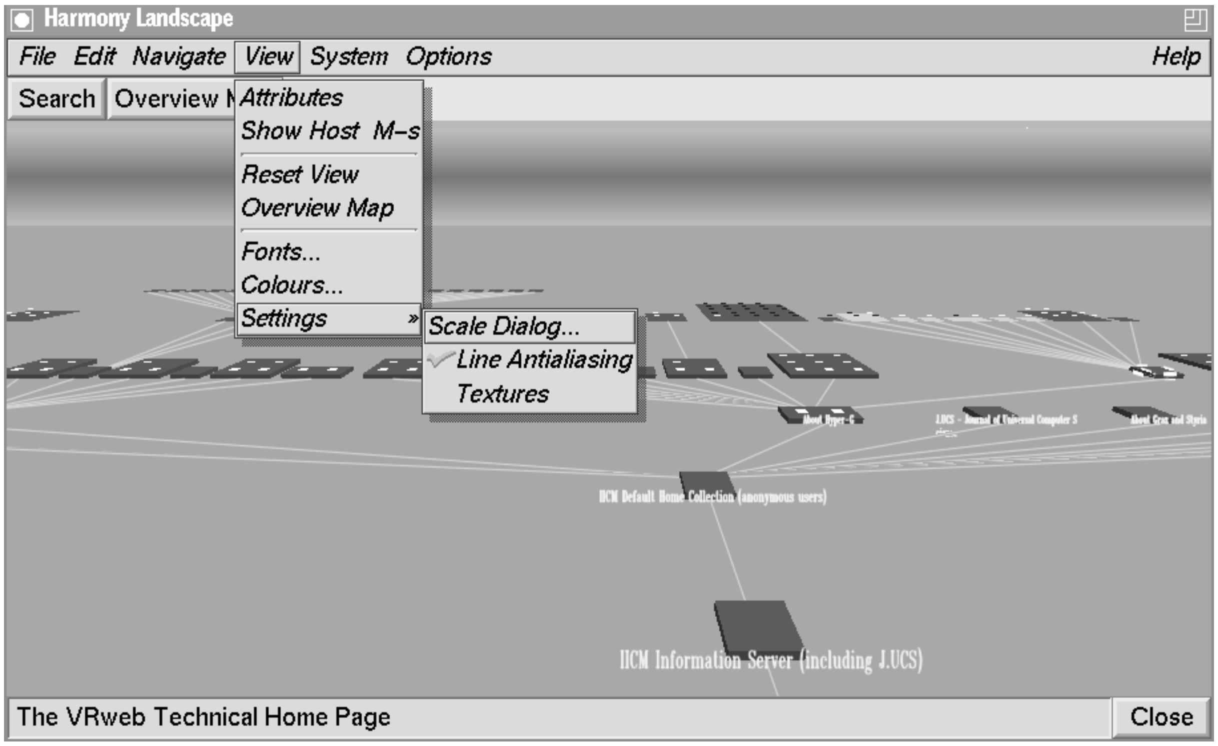

Harmony Landscape
File Edit Navigate View System Options Help
Search Overview M
Attributes
Show Host M-s
Reset View
Overview Map
Fonts...
Colours...
Settings »
Scale Dialog...
Line Antialiasing
Textures
IICM Default Home Collection (anonymous users)
IICM Information Server (including J.UCS)
The VRweb Technical Home Page
Close

Figure 5.5: Pull-Down Menu

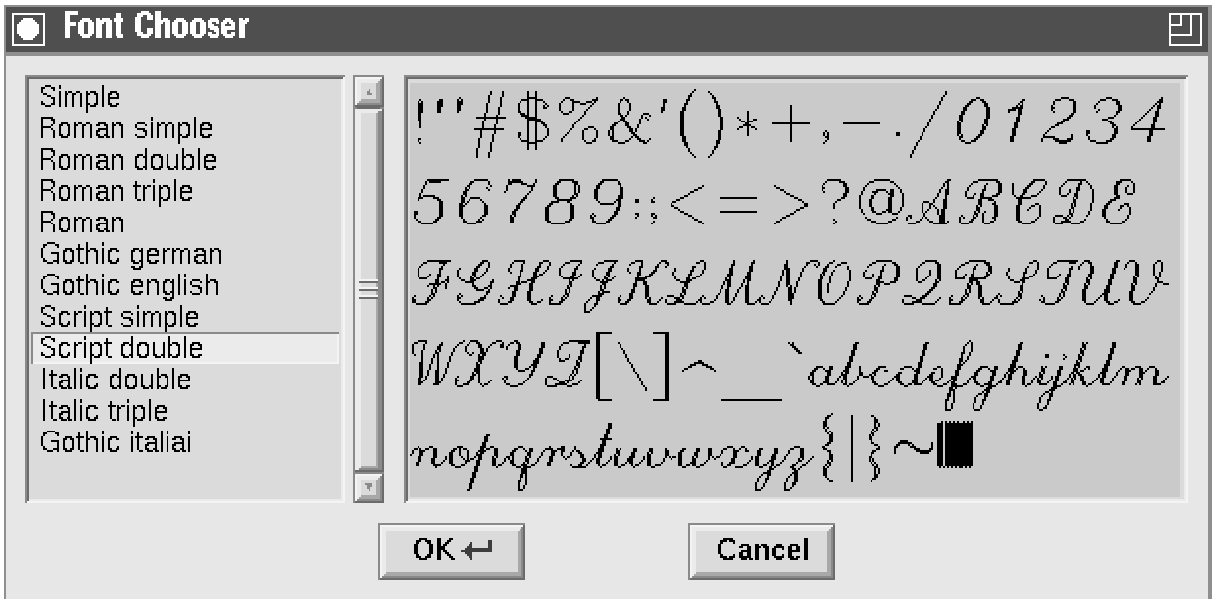
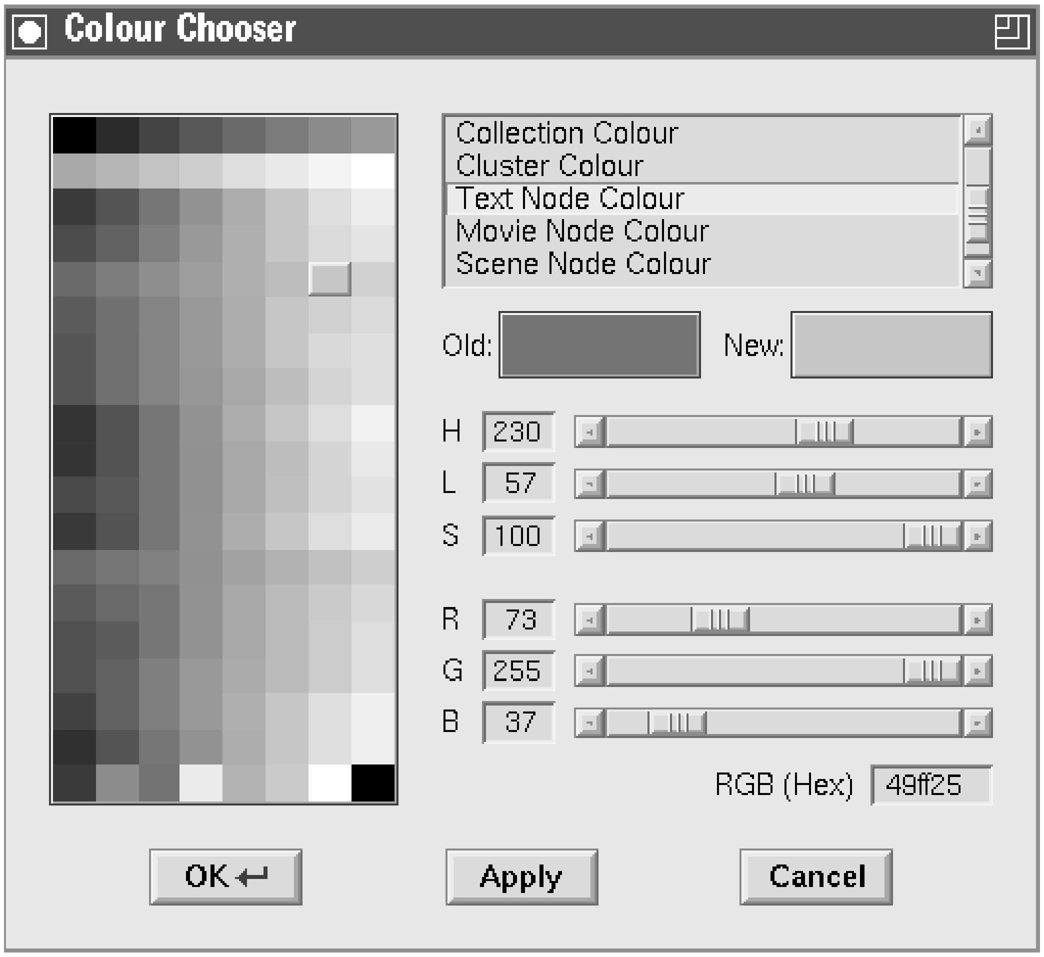


Figure 5.6: The Font Chooser



H 230
L 57
S 100
R 73
G 255
B 37
RGB (Hex) 49ff25

Figure 5.7: The Colour Chooser

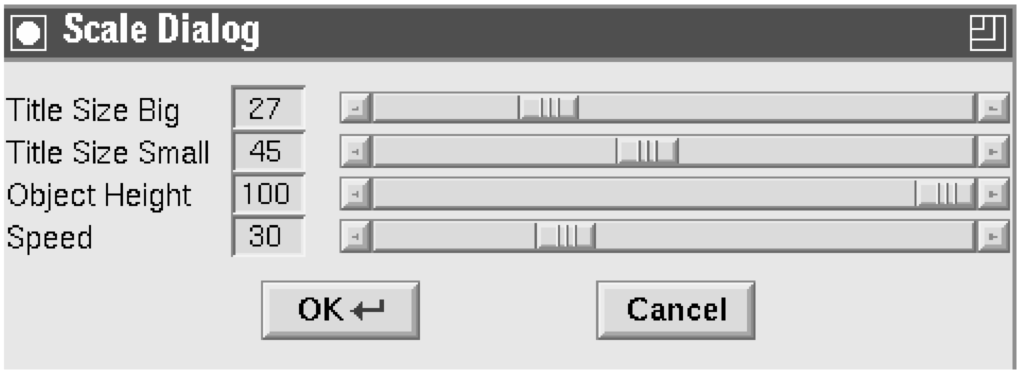
on the font name in the list and double-clicking displays the font in the window.

5.5.2 Colour Chooser

The colours of the Landscape can be changed by using the Colour Chooser (Figure 5.7). One chooses the desired object in the list and the desired colour. Clicking on the Apply button causes the new colours to be applied in the main Landscape window. Clicking on the Ok button commits the current colour settings, clicking on the cancel button exits the colour chooser without changing any colours.

5.5.3 Scale Dialog

The height of the document blocks, the size of the document titles (small) and the collections and clusters titles (big) can be modified with the scale dialog (Figure 5.8). The user can also scale the speed of all movements (flying, point of interest movement or animation) by using the fourth scrollbar.



Title Size Big 27
Title Size Small 45
Object Height 100
Speed 30

Figure 5.8: The Scale Dialog

5.6 Lettering

The block in the Landscape represents a document (collection or cluster) and the attributes and properties of the object are used to inform the user about the contents of the document. The height of the block corresponds to the size of the document and the colour to the type. But the size and the type of a document is not enough information. The user needs more information about the contents otherwise the collection tree is not useful. Hence a title is necessary, which is a very short description of the contents. In the Landscape the title is displayed in front of the object and so the user can have an idea of the contents.

For efficiency, vector fonts are used rather than raster fonts. The basic command for drawing a title is the "line command" (to draw a line from a starting point to a final point). The number of lines per character and the starting and final points of the lines are stored in a file and are loaded after starting the program. At the beginning of my thesis work I created a very simple font (Figure 5.9) with few lines per character and I developed a file format in order to store the data of the characters. The german special characters and two icons are also part of this font (the Eyl Simple Font). The icons are drawn at the top of a document in order to show the user the type of the document (text or image). I used a drawing program to draw the icons and stored them in the CGM file format. A program converted the CGM file format to this file format. Later we found out about the Hershey fonts (see also appendix C). They were taken over and modified a little. The Eyl Simple Font have been converted to the new Hershey file format and now several different fonts can be chosen by using the Font Chooser (see also Section 5.5.1). The Eyl Simple Font has an advantage over the other fonts. It is a very simple font and hence the drawing of the titles takes less time and the drawing cost can be reduced. It also supports german special characters.

The list with the font names in the font chooser is stored in an information file. The data of all fonts are stored in a single main file. In addition to the two files there are files for each font storing the information where the font data in the main file can be found. The names of these font files are also stored beside the font names in the information file.

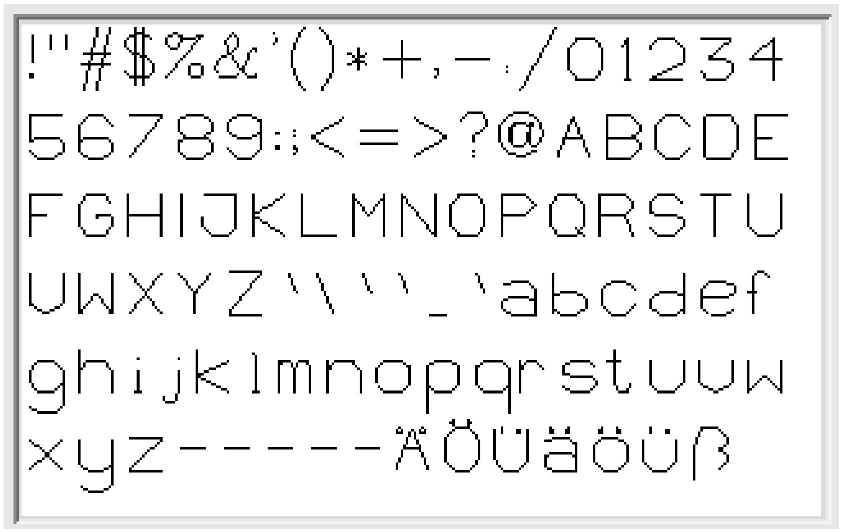


Figure 5.9: The Eyl Simple Font

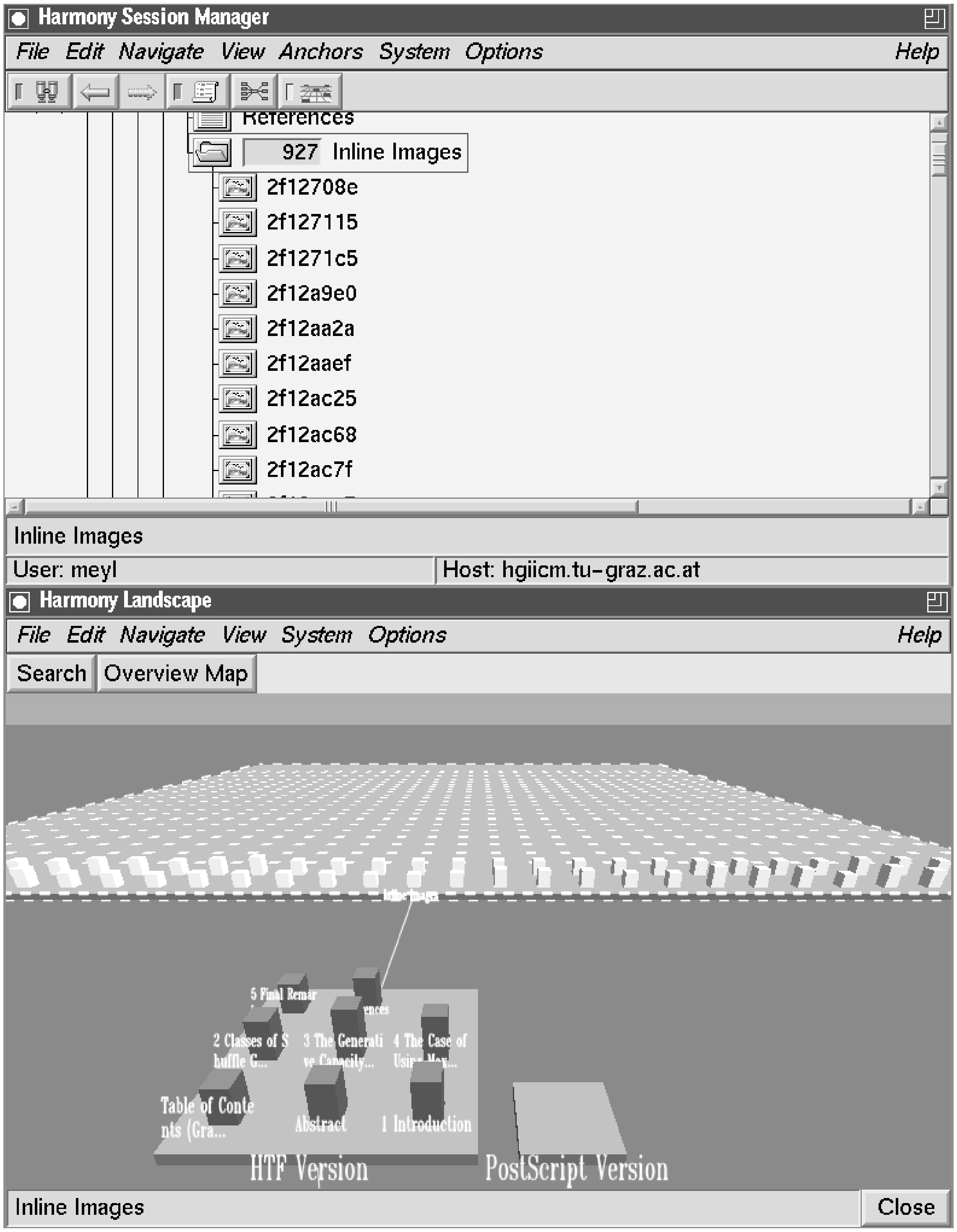
One can add an additional font or can delete a font by editing the information file. The names of the information file and the main data file can be changed by changing the X attributes.

The writing should be readable independent of the view position or the view direction. When an object is far away the lettering is drawn with a small line width and when the object is near it is drawn with thicker line width. The line of sight always strikes perpendicular to the letters and so the user always looks at the writing optimal. When the user change the view direction (between vertical and horizontal) the writing is also rotated (between horizontal and vertical). A proportional writing is used in order to display the title as long as possible and to use the existing space optimal.

5.7 Coupling with Collection Browser

The Landscape is part of the collection browser which displays the collection tree in a 2-dimensional list. The title and an icon symbolizing the type of the object are placed one beneath the other and are connected by lines to show the hierarchy. One can open and close the collections by double-clicking. When the user opens the Landscape by pressing the button "Landscape" in the window the same current collection tree is displayed in the Landscape. And so the Landscape is only another or additional way to view the tree. Users can decide which presentation they prefer. Depending on the task, intention of the user, or the size of the collection tree the collection browser or the Landscape is more suitable. A collection which contains a large number of documents can be represented more clearly in the landsape than in the collection browser. For example, Figure 5.10 shows a collection with a J.UCS paper and more than 900 inline images are part of this paper.

The collection browser and the Landscape are coupled (see Figure 5.11). When a collection is opened (closed) in the collection browser, the same collection is automatically



User: meyl

Host: hgiicm.tu-graz.ac.at

Figure 5.10: Landscape and Collection Browser with a large Collection

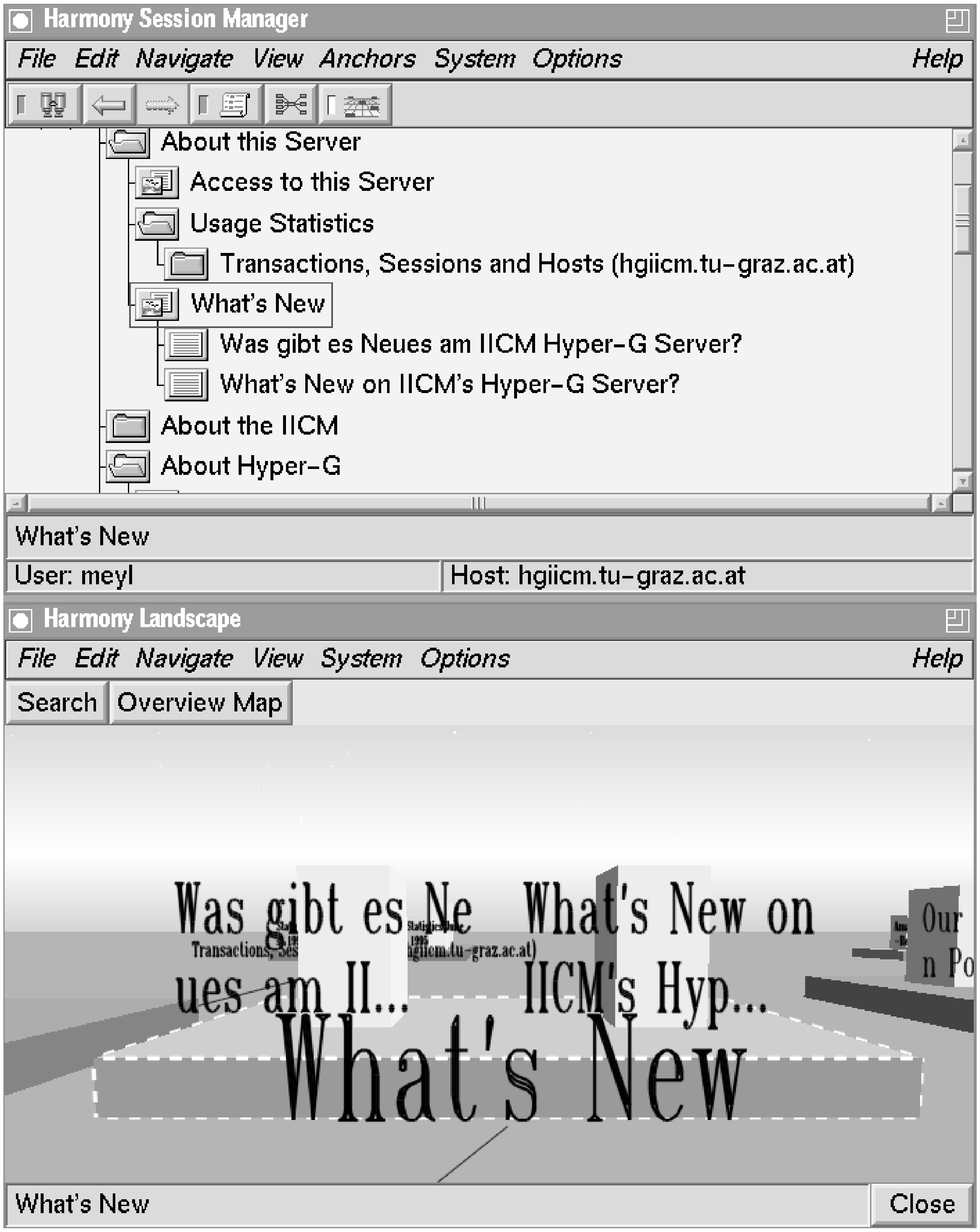

Harmony Session Manager
File Edit Navigate View Anchors System Options Help
About this Server
Access to this Server
Usage Statistics
Transactions, Sessions and Hosts (hgiicm.tu-graz.ac.at)
What's New
Was gibt es Neues am IICM Hyper-G Server?
What's New on IICM's Hyper-G Server?
About the IICM
About Hyper-G
What's New
User: meyl
Host: hgiicm.tu-graz.ac.at
Harmony Landscape
File Edit Navigate View System Options Help
Search
Overview Map
Was gibt es Neues am II...
What's New on IICM's Hyp...
What's New
What's New
Close

Figure 5.11: Coupling Collection Browser and Landscape

opened (closed) in the Landscape and conversely. So the trees always have the same contents and extension and also the current objects in the Landscape and in the collection browser are always the same.

5.8 Methods for Reducing the Cost of Drawing

The impression of movement is achieved by changing the position of the viewpoint and calculating and drawing the scene with the new viewpoint. Drawing is done in real-time and at least 20 frames per second are necessary for a smooth animation. When the number of objects in the scene increases, the time, which the computer needs to draw a scene, increases and the frames per second decreases. The collection tree can grow and grow during a session and so the Landscape has to present more and more blocks and characters. Therefore it is important to keep the cost of drawing as low as possible:

- One can easily determine the objects (blocks with lettering), which are behind the user and are not visible. These objects must not be taken into account for drawing.

- The Landscape calculates a perspective view. A few nearby objects appear large and those further away appear smaller and smaller. The title of a very small object is not readable and it is not necessary to draw this title. When a user flies towards an object and approaches within a certain distance the Landscape blends in the lettering. Unnecessary drawing is avoided.

- The size of a document is mapped to the height of a block. But it takes more time to draw a block than to draw a square (a block with the height equal to zero) on the plane. So all collections, clusters and documents which are within a certain distance from the viewpoint are represented by a block, objects further away are only represented by a square on the plane or on the collection (cluster). The distance depends on the height of the block that means that a very large document (very hight block) within a large distance and a small document within a small distance is displayed as block.

- When an object is highlighted (see Section 5.2) the Landscape has to draw the whole scene with this object newly. But using overlay bitplane the highlighted object can be drawn into this bitplane. The overlay bitplane is put on the top of the standard bitplane with the collection tree. And so objects can be highlighted fast without redrawing the collection tree. The cross in the overview map which represents the current position of the user in the main window can also be drawn in the overlay bitplane. When the user moves in the main window, the Landscape clears the overlay bitplane and draws the cross at the new position. The standard bitplane with the collection tree only has to be redrawn if the collection tree is changed. Hence the collection tree in the overview map must not be redrawn when the user moves in the main window.

  The GL (Silicon Graphics' native graphics library) supports overlay plane but OpenGL not and so a solution for OpenGL is necessary and has to be implemented.

- An object nearby the viewpoint should appear in front of an object, which is far away, independent of the succession of drawing and hidden surfaces should be eliminated. The z-buffer undertakes this task. High-performance graphics workstations have z-buffers in hardware and are very fast. In a computer without this hardware the software has to accomplish this task. When the Landscape runs on a computer without the hardware it disclaims the z-buffer (hence drawing is much faster). For that reason the Landscape has to draw the objects and the lettering in a fixed order. The succession of drawing is clear because the staring view direction is always straight ahead (see also Section 5.4). The following succession has been considered:

  1. From the horizon to the viewpoint
  2. From left side to the viewpoint, then from the right side to the viewpoint at each level
  3. When the viewpoint is higher than the collection or the cluster, first the document blocks and then the parent collection or cluster. When the collection or the cluster is higher than the viewpoint, first the parent collection or cluster and then the document blocks.
  4. First the block and then any lettering

This issue is becoming less and less important, because the prices for powerful graphic hardware are falling.

Chapter 6

Implementation

In this chapter I am going to introduce you to the classes (the program is written in C++) which comprise the Landscape. The parent classes of these classes are explained too. Certain selected details of the implementation, like picking an object or calculating the tree layout, are also described more fully. At the end of the chapter there is a section about possible extensions to the Landscape.

6.1 About the Implementation

The Landscape is part of the Session Manager and is not an independent process or program. The names of the source files end with "3d" for example "overview3d.C" and "overview3d.h" and so one can distinguish between the source files of the Landscape and the files of the Session Manager. The Landscape was implemented using C++ [30] under the UNIX operation system with the X11 window system.

The user interface was implemented by using the InterViews toolkit [17]. InterViews provides classes for menus, buttons, scroll bars, labels, input handlers and so on. These elements can be combined to build a user interface. In the course of time, the Harmony team has developed innerous InterViews widgets for example a file chooser or a colour chooser. Some of these dialog boxes were utilized in the Landscape. InterViews also supports X attributes which are used to define style values (for example a font or window size) and different variables outside the program in a configuration file (see also Appendix B). Users can change the configuration file to specify their preferences. Using InterViews and the widgets a consistent look and feel for all Hyper-G applications is possible.

The device-independent GE3D graphics library is used for 3-dimensional graphical output. This library is an interface between the application program (the Landscape) and any graphic library which is available on the platform. One can port the Landscape to another platform (graphic library) without changing the source of the Landscape. Only the GE3D library has to be adapted. At present the GE3D library is available for

- GL - Silicon Graphics' original native graphics library
- OpenGL - standardized 3D graphics interface which evolved out of GL and is now endorsed by numerous companies and institutions

OpenGL becomes more and more the standard for 3D graphics and is already offered by many vendors. The Landscape runs on following platforms:

- Silicon Graphic UNIX workstations with GL or OpenGL
- UNIX workstation with X11 window system which supports OpenGL
- any UNIX workstation with X11 window system using the Mesa library

The Mesa library uses only standard X11 functions and the program needs no special hardware for graphic output. Of course only relatively simple scenes can be drawn sufficiently fast and interactive movement is only possible when the collection tree is small.

The GEContext is the link between InterViews and the 3D graphic output. It is responsible for opening, placing and resizing the window for graphic output within the application window. GEContext must be adapted for different window systems.

6.2 The Main Class - View3D

The main class of the Landscape is the class View3D which is derived from the class Graph. The class Graph was originally implemented for the collection browser and is responsible for storing a graph, calculating a layout, presenting the graph to the user and so on. For example this class is used for showing the 2-dimensional collection tree. The class View3D inherits this capabilities and provides additional functions for 3-dimensional presentation. Some functions of the class Graph are overloaded for example the function 'draw' which is responsible for drawing the collection tree. The class Graph undertakes a lot of work and so I am going to introduce you to the parts of the class which are important for the class View3D. The class Graph accomplishes two basic tasks:

- caching the collection hierarchy from the Hyper-G server
- managing the collection hierarchy data

The collection hierarchy (collection tree) is stored in two lists:

- node-list
- edge-list

There are also index lists which are used to find a particular edge or node. Nodes and edges are classes. A node can be a collection, a cluster or a document (text, image, movie, ...) and the nodes are numbered consecutively. The edges link the parent node (collection or cluster) to the child node (subcollection, cluster or document). They are stored in the edge-list and are also numbered consecutively. An edge contains two node numbers, the node number where the edge starts and the node number where the edge finishes (see Figure 6.1). The edge-index-list is used to find a particular edge without searching the whole list. When one wants to know the edge-numbers from (or to) a particular node one can use the edgeindexfrom-list (or edgeindexto-list). For example one wants to know the first edge, which goes out from the node with the number two. The edge-number can be found at the second position in the edgeindexfrom-list. This edge-number is for example four. At the fourth position in the edge-list is the desired edge. More edges which also

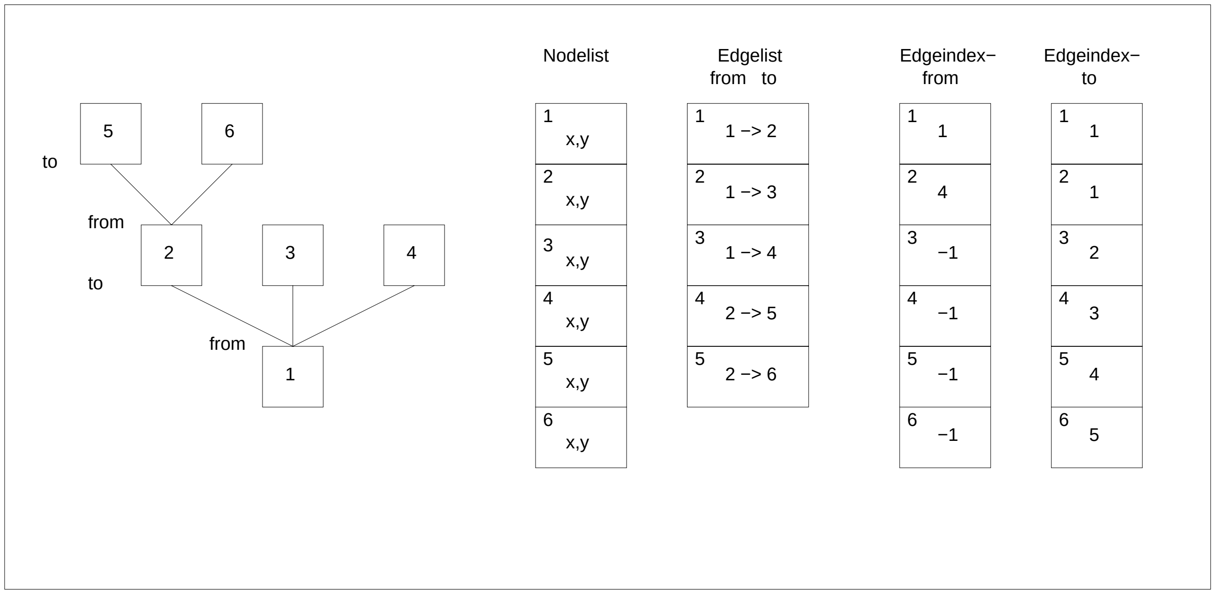


Figure 6.1: Node-list, Edge-list and Index-lists

go out from the node two are at the next positions in the edge-list, because this list is sorted by the node-numbers where the edge starts. Several functions provide access to these lists and are used in the class View3D. There are some function for caching nodes from the Hyper-G server for example 'takeAndInsertChildren' or 'insertNodeWithPath'. These functions are overloaded by the class View3D, because the metadata of the documents which are placed upon the collections must also be retrieved. Opening a collection caches all children. If the child is a collection or cluster it's children are also retrieved. The documents are placed upon the parent collection or cluster and the subcollections or clusters of the children are not displayed and remain invisible.

The class View3D is responsible for the 3-dimensional graphic output. This is a excerpt of the interface of the class View3D:

```
class View3D: public Graph
 {
  public:

   View3D (Sessionmanager* s,IvNode* root);
   ~View3D();

// request the desired geometry :
   virtual void request (Requisition&) const;
   virtual void allocate (Canvas*, const Allocation&, Extension&);
// draw the collection tree :
   virtual void draw (Canvas* c,const Allocation& a) const;
// catches the children of a node from the Hyper-G server :
   virtual int takeAndInsertChildren(IvNode*);
   virtual void insertNodeWithPath(IvNode* & ,IvNode* parent=0);
// calculates the layout of the collection tree :
   virtual void layout();
// determines the existing space for the lettering :
   void calcSpaceLetter(IvNode*,int);
// determines the picked object :
   void pickObject(float,float);
// starts an animation towards a node :
   void flyTo(
              IvNode*,            // the target node
// change view-direction and the height of the viewpoint or not :
              int nochange_z_alpha = 0);
   void flyToPoint(point3D p);     // point of interest movement
   void makeOverview();            // create the overview window
   virtual void handleKey(const Event& e); // handles key-events
// saves the current collection tree in a VRML file format :
   void saveAsVRML();

  private:
// InputHandler :
   virtual void keystroke(const Event&);     // pressed key
   virtual void move(const Event&);          // mouse movement
   virtual void press (const Event&);        // pressed mouse button
// double-c. mouse button :
   virtual void double_click(const Event&);
// mouse movement and button pressed :
   virtual void drag (const Event& e);
// released mouse button :
   virtual void release (const Event&);
   Camera cam_;                        // camera of the Landscape
   Overview* ov_;                      // overview window
};
```

The class View3D accomplishes following basic tasks:

- View3D calculates the tree layout and determines the position of the nodes. It also calculates the space between left and right neighbours for the lettering.
- The class stores the current viewpoint and line of sight in the class Camera and draws the horizon and the sky. It also calls the 'draw' function of the nodes and draws the edges between the collections or clusters on the plane.
- It determines the object which is beneath the mouse cursor.
- The class View3D is derived from the class Inputhandler which is responsible for events (for example a mouse movement or a pressed key). The Inputhandler becomes

aware of an event and calls the appropriate member function of the class View3D. In this function the Landscape can react on the event. So the user can open or close collections, view a document, select an object or fly over the landscape.

- It provides different navigation modes.

6.2.1 Tree Layout

Some layout algorithms are implemented in class Graph. However, these algorithms are not satisfactory because they are too slow or the layout is unsuitable. So I developed a very simple algorithm for a symmetrical tree layout. The algorithm uses a recursive function with a node as parameter. The function is called with the root node. In the function the same function is called with the children as parameter and then the parent node is placed. For example a tree has three levels (root-children-grandchildren). The algorithm starts with the grandchildren and places them side by side. The children are placed symmetrically above them. When two children overlaps one child has to be moved with its children (grandchildren). Finally the root node is placed symmetrically above the children.

6.2.2 Picking 3D Objects

When the user clicks on a point on the window the program has to determine which object in the landscape was hit (see Figure 6.2). The program needs this picking function for highlighting, selecting and activating an object (open or close a collection, view a document or a cluster). Picking is also necessary for point of interest movement. The input of the picking algorithms are the x-y window coordinates of the hit point and the current camera settings (the current perspective projection). Several steps are necessary to determine the hit object or hit point in world coordinates [24]:

1. Calculating 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. In the picking algorithms A is the viewpoint and b is the direction from the viewpoint to the hit point in world coordinates or the direction from the viewpoint to a point on the view-plane which correspond to the hit point on the window in normalized window coordinates. t must be greater than or equal the distance of the near clipping plane.

2. Searching for objects which the ray hits
   All objects in the landscape are cubes and so a fast algorithms can be used for testing. More complicated algorithms have to be used for an object with arbitrary faces. The ray can hit several objects but the object which is nearest the viewpoint or with the minimum value of t is the right one.

3. Calculating the hit point
   The hit point is given by $A + thit * b$ and is used for the point of interest movement or for picking the edges. If the ray hits no object the point where the ray intersects the plane is calculated. Each edge is tested with this point.

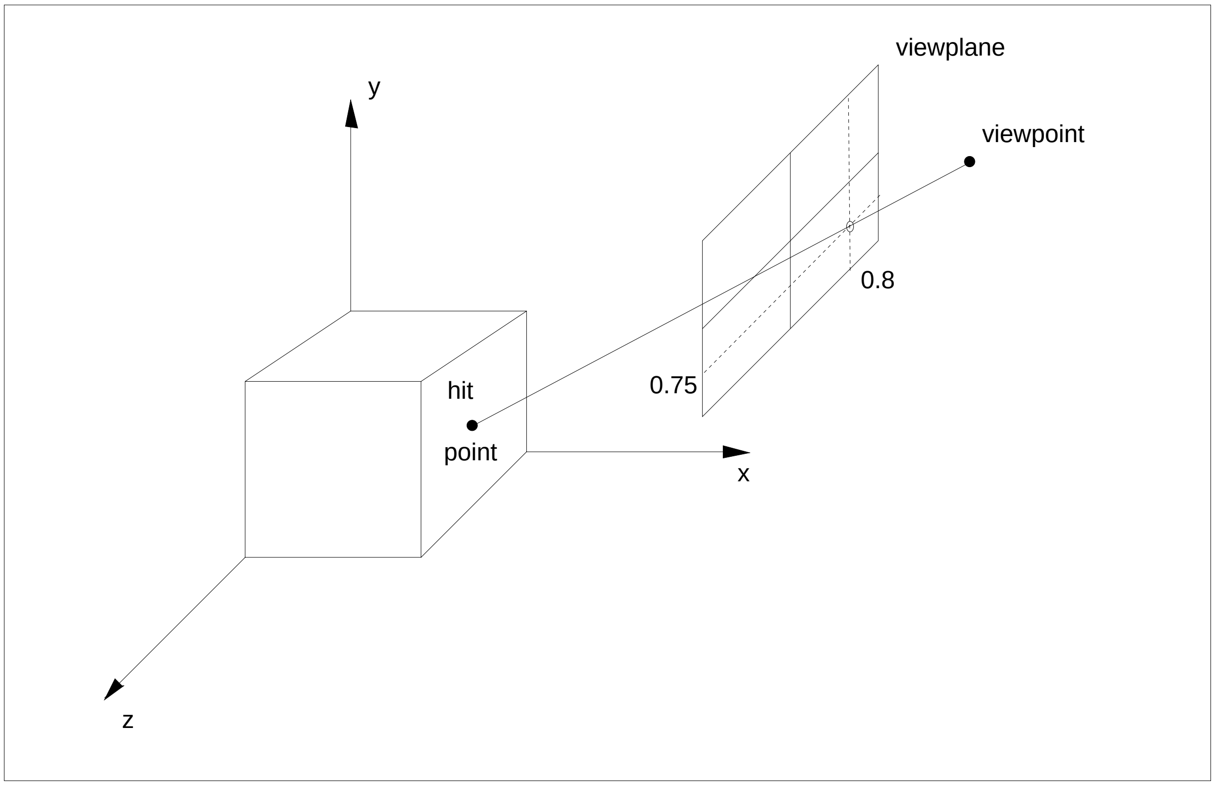


Figure 6.2: Picking an Object

6.2.3 Movement

Movement is achieved by changing the position of the camera and the line of sight can be changed by changing the lookat point. The position of the camera and the lookat point are stored in the class Camera. The speed of movement is equal $k * x$. x is the distance the mouse was dragged as fraction of the window size and k is a factor which the user can change (see therefore section 5.5.3).

The speed of the movement towards the point of interest (POI) is proportional to the current distance from it and so the distance from the POI decreases (or increases) with a logarithmic function. The approach to the point is fast at the beginning but a collision never happens. The motion is stopped at the near clipping plane (otherwise the POI would disappear).

6.3 Node

Nodes and edges build up the collection tree and there are a lot of different types of nodes : collection, gophercollection, cluster and document nodes. A document node can be a text, image, movie, scene (3D), postscript, generic, sound, telnet, ftp, gophertext, gopherimage, gophersearch, gophermovie, gophersound, gophertelnet, wwwtext, wwwimage, wwwmovie, wwwscene, wwwsound, wwwpostscript or a wais node. Each type of a node is represented by a class which is derived from the class Node3D. Node3D is derived from the class IvNode which is again derived from the class Node. Node and IvNode were originally implemented for the Session Manager.

The class Node is the basic class and it stores the position of the node (x,y) and the title. It also stores information about the position in the database. The class IvNode provides capabilities for displaying the 2-dimensional node in an InterViews window. The class Node3D provides additional functions for 3-dimensional presentation. The following short excerpt of the class definition shows a part of the public interface of the class Node3D:

```
class Node3D : public IvNode
{
public:
  Node3D(const char* name,long id,Graph* =nil,float x=0,float y=0,
         float api=1,float vw=1,int level=-1);
 ~Node3D();

 virtual void draw(Canvas*, const Allocation&);   // draw the node
 virtual void drawText()const;                    // draw the title
 // draw a selected node with an additional wired cube:
 virtual void drawBorder(Canvas*, Color*, Brush*);

 virtual float xRequired()const; // the required space in x
 virtual float yRequired()const; // the required space in y
 virtual NodeTyps getType()const {return Text;}
 // test if the ray hits the node (for picking):
```

```
 virtual float rayHits(point3D&,vector3D&,float,float);
 virtual void setX(float x); // set position of the node (overloaded):
 virtual void setY(float y);
 // set the existing space for lettering:
 virtual void setSpace(float s);
 // determine the size of the document and calculates the height of
 // the object:
 virtual void getDocSize();
 // save node in VRML file format:
 virtual int saveVRML (FILE* );
};
```

During the layout calculation the position of the node and the space for the title is set. The member functions 'draw' and 'drawtext' are called for all nodes in the 'draw' function of the class View3D.

6.4 The Overview, GlText and Camera Classes

There are some more classes in the Landscape. This is a short description of the more important ones:

- Overview
  The class Overview is responsible for the flat overview window. The position of the overview camera is over the collection tree and one looks from above on the tree. The user can not change the position or the line of sight. Overview doesn't possess an own collection tree but it accesses the data in the class View3D using public member functions. The class accomplishes following tasks: drawing, picking and handling of user inputs at which key events are passed on to the class View3D.
- GlText
  This class loads and stores the starting and final points of the lines which build up the characters or letters. There are functions for drawing a single character or a whole string. The parameter of the function which draws a string are the position, the available space for the string, the size of the characters and the angle (which depends on the line of sight) of the characters. This function uses two lines for long titles.
- Camera
  The Landscape uses a perspective and untilted camera which is defined by a viewpoint (the position of the camera) and a reference point (or lookat point). The vector viewpoint - lookat point is the line of sight and is called n. The distance between viewpoint and view-plane is called focal length. The aperture is the height of the view-port and the aspect ration is the with/height ratio of the view-port. The up vector always equals (0,1,0) for an untilted camera. Other vectors of the

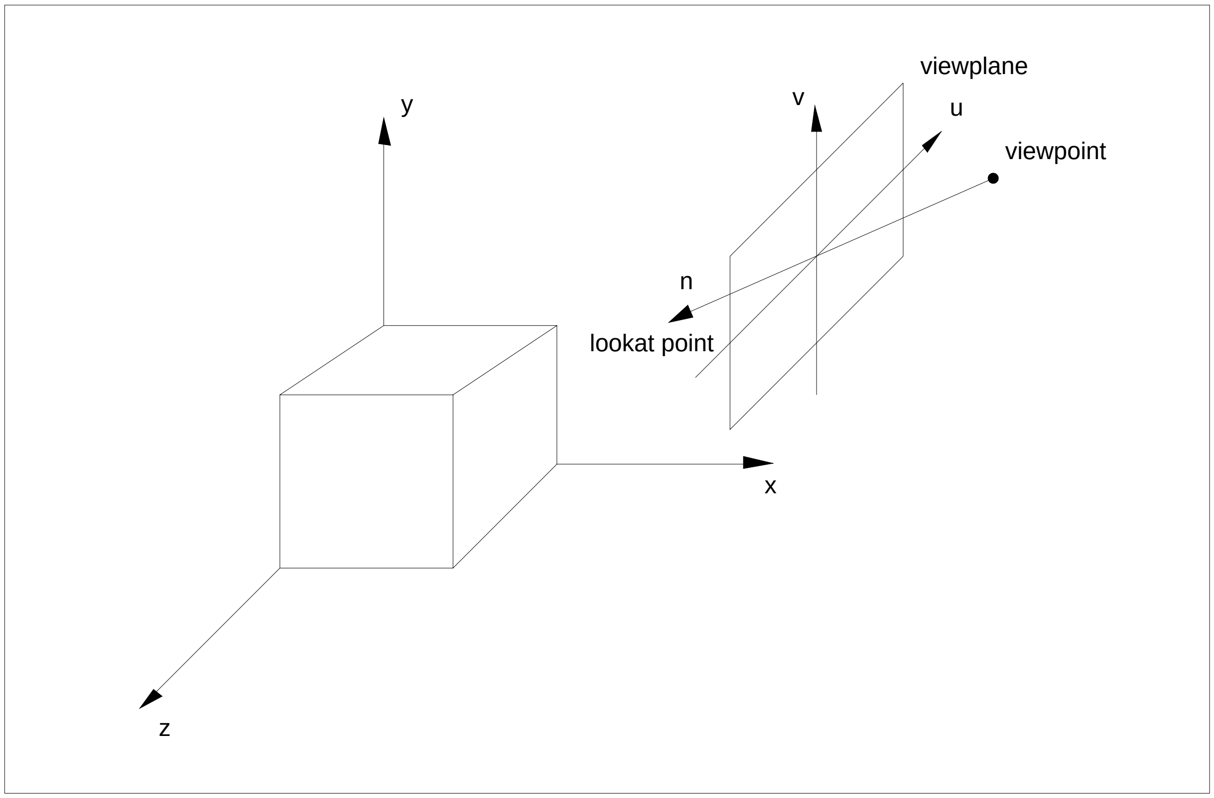


Figure 6.3: The Camera Model

camera are u and v and they can be calculate as follows: u = n x up (cross product) and v = u x n. The vectors u, v, n form an orthogonal coordinate system, u and v lie in the view plane horizontally and vertically (see Figure 6.3). The class camera3D stores the viewpoint, the lookat point, aperture, up vector, focal length and the aspect radio and there are public member function for changing the viewpoint and the lookat point. The aspect radio is set in the function allocate and it equals the with/height ratio of the window. The classes View3D and Overview possess an object of class Camera.

6.5 Outlook

The Landscape is not ready yet and more than a single Master's Thesis is necessary to use and implement all advantages of a 3-dimensional presentation of information. The hardware becomes more and more powerful and one will not be limited because of less computer power. These are some suggestion which maybe could in future be implemented:

- A 3-dimensional object has different attributes: shape, size, colour and texture. At present only colour and size are used but the other attributes should also be used to inform the user as much as possible about the contents of the object. Naturally the shape should be mapped to the type of a document. For example a cylinder could be a text document and a pyramid an image document. But the user has to remember which shape represents which type of document. If an object has the appearance of a book or a camera the user can know which document is represented without additional knowledge. Such an object is more complicated and more computer power is necessary.

The colour could be used for the age of an object or for something else. A texture could also present an attribute of the document or one could display a image or a icon at or upon the object by using texturing.

- Different techniques could be used to increase the realistic impression of a landscape for example mountains on the horizon or a gras texture on the plane.
- Drawing in the Landscape with the Mesa-library can be very slow. In that case interactive movement can be improved by fading out the lettering and/or by drawing the object in wireframe mode during movement. When the movement stops then the collection tree is displayed with lettering and in flat shading modus.
- At present the user can not turn around in the Landscape. One could lift this restriction for expert users (and retain it for novice users) and implement a user interface which is similar to the Harmony 3D Viewer. So expert users can move as they like.
- There are a lot of different constant values which determine the appearance of the landscape: the height of a collection and cluster, the size (width and not the height) of a document block, the distance between neighbouring documents, collections or clusters, the distance between two levels of the tree. These values could be modified by the user (in a dialog box with scroll bars or/and X attributes) and the user would determine the appearance of the landscape. The frame within which lettering is shown or a block is drawn could also be decided by the user.
- The Landscape only presents the collection hierarchy but no links between documents. The incoming and outcoming links associated with a particular document could be displayed using the 3rd dimension. Another possibility is to show the links in another 3-dimensional space similar to the local map which is a 2-dimensional map of incoming and outcoming links.
- In future, collision detection could be implemented and so one could not fly into an object or through an object because motion would be stopped. At present an object disappears suddenly when the viewpoint is too close.
- At the moment the Landscape only presents a view of the collection tree. In future it could also be used to modify the collection tree for example one could take, duplicate or create an object and put it to any desired place or into the waste bin.

Chapter 7

User Guide

This chapter is a short summary of the important commands and the handling of the mouse. User working with the Landscape the first time should consult this user guide.

The user can open and close the Landscape window by clicking on the landscape-button in the button row in the collection browser. At the beginning the collection tree with the home collection is shown (if the user have not navigated in the Session Manager first). When the user is not identified the anonymous home collection is presented. Different objects (blocks) are spread out in the landscape. One distinguishes between collections, clusters and documents. The document blocks are placed upon the collection-socles and cluster-socles. The type of an object (collection, cluster, text-document, image-document, ...) can be recognized by the colour. The user can change this colours by using the colour chooser (menu View - Colours) or by changing the X attributes.

Current Object
One object in the tree is always the current object. The edges of a current block are drawn in another particular colour and are dotted lines. The title of this object are displayed in the status line. Commands are executed with this current object for example pressing the '+'-button opens the current collection. One can change the current object by clicking on an object or on the line to the object with the left mouse button or by using the cursor buttons. The child (collection) of the current object becomes the current object by pressing the $\uparrow$ button, the parent object (collection) by pressing the $\downarrow$ button, the left neighbour by pressing the $\leftarrow$ button and the right neighbour by pressing the $\rightarrow$ button. Pressing the shift and $\uparrow$ buttons changes the current object to a document object which is placed upon the current collection and pressing the shift and $\downarrow$ buttons changes the current object to the collection object which is the parent of the current document object.

Activating an Object
One can activate an object by double-clicking on the object. If an open (closed) collection is activated then the collection will be closed (opened). If a cluster is activated then all documents which are part of this cluster will be displayed using the corresponding viewer or the text viewer will choose the document with the desired language and will present it. If a document is activated then the user can view this document.

Navigation

The user can move in the landscape by using the mouse. Motion is achieved by dragging the mouse while pressing a button. When a mouse button is pressed, the Landscape draws a cross at the position of the mouse cursor. The relative position of the cursor to the cross determines the direction of flight and the speed. The speed is proportional to the distance between the cross and the cursor and the direction from the cross to the cursor determines the direction of flight.

Another kind of motion is a smooth animation towards a desired object which for example can be selected in the result list of a search or in the history list by clicking on the list item. Table 7.1 shows the mouse button assignment. If the movement is too

Situation	*Hit Place*	*Mouse Button*	*Mouse Movement*	*Resulting Movement*
		left	up / down	flying forward / backward
		left	left / right	flying left / right
		middle	up / down	flying up / down
		middle	left / right	flying left / right
		right	up / down	tilt the head up / down
Ctrl button	any place	middle		movement towards POI
Ctrl button	any place	right		movement away from POI
double-click	object	right		animation towards object
double-click	line	right		a. towards object at the end of the line.
double-click	object	middle		animation towards parent
during animation		right		stop animation
during animation		middle		jump immediately to final destination

Table 7.1: Mouse Button Assignment

slow or too fast one can change the speed by changing the speed value in the scale dialog (menu View - Scale Dialog - Speed).

Overview Map

The user can open a window with an overview map of the current collection tree by clicking on the button 'Overview' in the landscape window or by selecting the menu item View - Overview. A red cross in this map shows the current position of the user in the main window. The overview window supports an additional kind of navigation. One can select any place in the overview window by moving the mouse cursor to the desired place and pressing the left mouse button. Then the viewpoint in the main window moves towards this place and the cross in the overview map moves towards the mouse cursor.

Commands

Several commands can be executed by pressing a particular button. Clicking on an object with the right mouse-button opens a window with information about this object (type, title , access rights ...).

Button	*Command*
Space	fly towards the current object
+	opens current collection
-	closes current collection
c	inserts children of current collection or cluster
Ctrl-c	closes the whole collection tree
↑↓←→	changes current object
Return	activates the current object
s	starts the search-dialog
h	shows history-list
l	shows local map of current object
S	stores the current collection tree as VRML

Table 7.2: Commands

Chapter 8

Concluding Remarks

Over the past 30 years user interfaces have been improved enormously because computer performance has increased. In the beginning computer users had to be computer experts and they had to work with punched cards and long computer printings. Today more and more research projects and programs take advantage of spatial metaphors. The 3-dimensional presentation of large amounts of information has numerous advantages. The Harmony Information Landscape is the first attempt in in Hyper-G to use a spatial metaphor. It represents hierarchically structured information stored on a Hyper-G server. Hyper-G is a large-scale, networked hypermedia system and allows access to heterogenous information. Documents are connected by links and one can browse through the information web as one likes. The system provides an additional hierarchical structure which allows the user to maintain orientation in the large information web. The Landscape shows this hierarchical structure as an open landscape. Objects representing collections, clusters and documents are spread out on a plane and one can interactively fly over the virtual landscape.

Of course, a single Master's thesis can only begin to explore the possibilities of three-dimensional information visualisation. No hyperlinks are shown in the landscape and all objects in the landscape are blocks. The system-readable attributes of a document, cluster or collection could be mapped to the properties of an object (colour, size, shape or texture). At the moment the colours of the objects show the type of the document, but one could also use the shape of an object to represent its type: for example a camera can represent an image and a book a text document. A creased and worn book could represent an old text and so on. Users could recognize the type and other attributes of an object at a glance.

Other features which could also be implemented include interactively changing the hierarchical structure. The 3rd dimension could be used to present information on different levels or/and to display hyperlinks and one could also support the use of sound as part of a representation for a place or object.

At the moment, immersive 3D devices (such as data glove and data helmet) are not widely used, but they have some important advantages. The data helmet can track the orientation of the head and change the orientation of the viewpoint and it gives the impression that one is really part of the landscape. One only has to point at the place where one wants to fly. In future one could use such devices enabling users to walk and fly around in a virtual information world and look for interesting information. They will be able to arrange part of this world as they like. One might be able to ask the computer questions and be flown to the desired place. The current Landscape should be thought of as an initial proof-of-concept prototype, with many possible extensions still to be investigated.

The Harmony Information Landscape is currently available, for most common UNIX workstations, with software-only graphics and with hardware-accelerated graphics for platforms with OpenGL support.

Appendix A

Colour plates

Appendix B

X Resources

Several values and settings can be changed by editing the file with the X attributes. This is a list of all X attributes which concern the Landscape:

```
!! the position and the size of the Landscape window.
Harmony.Session.Landscape.geometry:   1200x600+40+170

!! scaling factor for the height of the document blocks [0..100]
Harmony.Session.Landscape.DocumentHeight: 50

!! scaling factor for the size of the titles [0..100]
!! the size of the collection titles and cluster titles
Harmony.Session.Landscape.TitleSizeBig:   27
!! the size of the document titles (which are located upon
!! the collections and clusters)
Harmony.Session.Landscape.TitleSizeSmall: 45
!! speed factor for all movements [0..100]
Harmony.Session.Landscape.Speed: 30

!! the name of the main font file with all data
Harmony.Session.Landscape.fontdeffile:          3dfonts/hersh.oc
!! the file name of the current font
!! this file contains the information where the data (for one font) in
!! the main file can be found
Harmony.Session.Landscape.fontfile:             3dfonts/simple.hmp
!! the name of the information file which contains the font names and
!! file names of all existing fonts
Harmony.Session.Landscape.fontinfofile:         3dfonts/fontinfo.dat

!! colours
Harmony.Session.Landscape.planecolour:              #009500
Harmony.Session.Landscape.skycolour:                blue
Harmony.Session.Landscape.edgecolour:               blue
Harmony.Session.Landscape.highlightedcolour:        white
Harmony.Session.Landscape.iconcolour:               black
```

```
!! edge colour of the current object
Harmony.Session.Landscape.selectedcolour:           red
!! colour of all titles
Harmony.Session.Landscape.textcolour:               black

Harmony.Session.Landscape.Collection.colour:        #50ff50
Harmony.Session.Landscape.GopherColl.colour:        #00ff00
Harmony.Session.Landscape.Cluster.colour:           cyan
Harmony.Session.Landscape.Text.colour:              #b000e0
Harmony.Session.Landscape.Image.colour:             blue
Harmony.Session.Landscape.Scene.colour:             #ffff00
Harmony.Session.Landscape.Movie.colour:             #00ffff
Harmony.Session.Landscape.Sound.colour:             #00ff60
Harmony.Session.Landscape.Generic.colour:           #90ff20
Harmony.Session.Landscape.Drawing.colour:           #ff00ff
Harmony.Session.Landscape.Remote.colour:            #30a050
Harmony.Session.Landscape.Telnet.colour:            #10a020
Harmony.Session.Landscape.PostScript.colour:        #00a020
Harmony.Session.Landscape.GopherNode.colour:        #b000e0
Harmony.Session.Landscape.GopherText.colour:        #b000e0
Harmony.Session.Landscape.GopherImage.colour:       blue
Harmony.Session.Landscape.GopherMovie.colour:       #00ffff
Harmony.Session.Landscape.GopherSearch.colour:      #909010
Harmony.Session.Landscape.GopherSound.colour:       #909020
Harmony.Session.Landscape.GopherTelnet:             #909030
Harmony.Session.Landscape.WWWNode:                  #909090
Harmony.Session.Landscape.WWWText.colour:           #b020c9
Harmony.Session.Landscape.WWWImage.colour:          #b010c5
Harmony.Session.Landscape.WWWMovie.colour:          #b000c2
Harmony.Session.Landscape.WWWSound.colour:          #b015c0
Harmony.Session.Landscape.WWWPostScript.colour:     #a015c0
Harmony.Session.Landscape.WWWScene:                 #a01510
Harmony.Session.Landscape.FTP:                      #f01510
Harmony.Session.Landscape.WAIS:                     #f000f0

!! sunset on : sunset and sunrise during a session
!! sunset off: the colours dosen't change
Harmony.Session.Landscape.sunset : on
!! nosunsetbackground on : the sky is displayed in the flat shading
!! mode with the colour 'skycolour' (see above)
!! nosunsetbackground off: smooth shading sky
Harmony.Session.Landscape.nosunsetbackground : off
Harmony.Session.Landscape.lineantialiasing : off
Harmony.Session.Landscape.texturing : off
!! Using the Landscape with MESA library nosunsetbackground should be
!! on, lineantialiasing and texturing should be off
!! texture for the plane
Harmony.Session.Landscape.upperbackgtexture: textures/skytxc.tif
!! texture for the sky
```

```
Harmony.Session.Landscape.lowerbackgtexture: textures/grasstxc.tif
Harmony.Session.Landscape.Collection.texture: textures/marmor128.tif
Harmony.Session.Landscape.Cluster.texture:   texture/stone.tif
```

Appendix C

The Hershey Fonts

The Hershey vector Fonts were originally created by Dr. A. V. Hershey while working at the U. S. National Bureau of Standards. The format of the font data was originally created by James Hurt. For the Harmony Information Landscape, the fonts were converted into a slightly modified format. The data of all fonts are stored in a main data file and each character or letter is stored as a record. Lines build up a character or a letter and so a record contains the points (x,y coordinate) of these lines. For example, record with the number 2 has the entry :

00002016MWOMOV ROMSMUNUPSQ ROQSQURUUSVOV

The first five characters are the record number (2), and the following 3 are the number of character pairs (16) which are following and so there are 32 additional characters. Each pair is the x and y coordinate of a point. The characters are encoded relative to 'R' which is zero (Q = -1, S = 1, and so on). The first pair is a left and right offset for proportional spacing. The second pair is the first point of a line and the third pair is the second point and so on. The character SPACE in the x coordinate means that a new line begins and the next pair is the the starting point of the new line and the previous pair is the last point of the old line. Thus the line above encodes:

record 2 (16 pairs or 32 characters)

```
spacing (-5, 5) MW
skip    (-3,-5) OM
draw    (-3, 4) OV
new line        R
skip    (-3,-5) OM
draw    ( 1,-5) SM
draw    ( 3,-4) UN
draw    ( 3,-2) UP
draw    ( 1,-1) SQ
new line         R
skip    (-3,-1) OQ
draw    ( 1,-1) SQ
draw    ( 3, 0) UR
draw    ( 3, 3) UU
draw    ( 1, 4) SV
draw    (-3, 4) OV
```

Most special symbols were deleted from the original main data file because the Landscape doesn't need them. Each record (or character) has a unique number (the first number in the record) and the record numbers of all characters which belongs to a particular font are stored in a separate font file (for instance simple.hmp or script.hmp). Here is an exerpt from the file italicc.hmp:

```
32 2199 0
33 2764 0
34 2778 0
35 733 0
36 2769 0
37 2271 0
38 2272 0
39 2777 0
40 2771 0
41 2772 0
42 728 0
43 725 0
44 2761 0
45 724 0
46 2760 0
47 720 0
48 2750 2759
58 2762 2763
97 2151 2176
```

The first number is the ASCII value of the character and the next number is the number of the record in the main data file for example the character with the ASCII value 36 is stored in the record with the number 2769. The record numbers of several characters can be determined with a third value for example the characters with the ASCII values from 48 to 58 are stored in the record numbers 2750 to 2759.

A new font can be added by appending the data (with unique numbers) in the main data file and creating a new file with the ASCII values and the record numbers. The name of this file and the font name must be append in the information file and then the new font can be chosen using the font chooser.

One can determine the names of the main data file, the information file and the current font file by editing the X attributes:

```
Harmony.Session.Landscape.fontdeffile:          3dfonts/hersh.oc
Harmony.Session.Landscape.fontinfofile:         3dfonts/fontinfo.dat
Harmony.Session.Landscape.fontfile:             3dfonts/romanc.hmp
```

For more information about the Hershey font see the Hershey distribution:
ftp://gondwana.ecr.mu.oz.au/pub/hershey.tar.gz

Appendix D

Graphics Engine 3D (GE3D)

This section is an updated version of Appendix B of Michael Pichler's thesis [24]:

The GE3D library - Graphics Engine 3D - was designed as a machine independent, immediate mode, 3D graphics interface. The first version of GE3D was developed together with Michael Hofer, whose work is acknowledged here. The functionality of GE3D includes:

- manipulation of vectors and matrices, and a stack of transformation matrices
- camera definition, both perspective and orthographic
- definition of light sources
- double buffering (two screen pages)
- drawings in wire frame, hidden line, flat shaded and smooth shaded
- drawing of 3D faces (polygons) and polyhedra
- drawing some 2D primitives: lines, rectangles, arcs, circles, output of text

The library was implemented in (standard) C atop the GL graphics library of Silicon Graphics and OpenGL. Header and implementation file contain macros to switch between ANSI C and Kernighan-Ritchie C by defining the preprocessor symbol GE3D_PROTOTYPES.

As it can be seen from the list above, GE3D's functionality is on a higher level than typical machine dependent libraries like GL or OpenGL. For example a polyhedron represents a set of faces with the same edge and fill colour. The drawing modes (from wire frame to smooth shading) do not bother the user of the library with correct settings of flags for hidden surface elimination, filling, usage of z-Buffer and so on. Other functions, for example for drawing lines and rectangles, and for manipulations of the matrix stack, have a corresponding counterpart in low level graphic libraries.

One could ask for the reason why implementing yet another graphics interface (moreover when it is closely related to GL). Beside the mentioned enlarged functionality the machine independence of the interface increases the *portability* of the programs. As the header file is completely independent of any other header file of graphics interfaces another implementation of the GE3D library can be linked at any time without changes or recompilation of existing programs.

D.1 Type Definitions

File <ge3d/vectors.h> contains the definitions for 3D *points* and *vectors* and a set of pre-processor macros for vector operations.

typedef struct
{ float x, y, z;
} **vector3D**, **point3D**;

typedef float **matrix** [4][4];

Type *matrix* is used for transformations (4D homogeneous coordinates). There is no assumption on the ordering of elements in the matrix (row major or column major). Functions of GE3D should be used to build and to concatenate the matrices, they can be stored and pushed onto the transformation stack, but should not be manipulated.

The file <ge3d/color.h> contains the definition for colours. The colour components R, G and B are in range 0.0 to 1.0.

typedef struct
{ float R, G, B;
} **colorRGB**;

The type *face*, specifying a polygon in 3D space is defined in file <ge3d/face.h>. See procedure ge3d_polyhedron for further explanation of fields.

```
typedef struct
{ int num_faceverts,    /* number of vertices in face */
      num_facenormals;  /* number of normals in face */
  int *facevert,        /* array of indices of face vertices */
      *facenormal;      /* ... and of the normals of the face */
      *facetexvert;      /* ... and texture vertices of the face */
  vector3D normal;      /* normalised, outward face normal */
} face;
```

The five supported *drawing modes* are defined in <ge3d/ge3d.h>:

enum **ge3d_mode_t**{ ge3d_wireframe,ge3d_hidden_line, ge3d_flat_shading,ge3d_smooth_shading,ge3d_texturing};

D.2 Functions

D.2.1 Opening the Graphics Device

There are two ways for opening the graphics device. Either GE3D is asked to open a window - this is done with the function

void **ge3d_openwindow** ();

Otherwise the main program is responsible for opening an output window. A call to

void **ge3d_init_** ();

will initialize the GE3D library, including clearing the screen and setting default values. A call of ge3d_open_window automatically causes ge3d_init_ to be called.

D.2.2 Display Control

void **ge3d_clearscreen** ();

clears the window (with the current background colour) and the z-Buffer (if hidden surface elimination is activated).

void **ge3d_swapbuffers** ();

The GE3D library uses double-buffering (if available). This means that there are two screen pages for graphic output: a *visible* and an *active* one. All drawings are done on the *in*visible page. ge3d_swapbuffers must be called to display a drawn picture by swapping the two pages. The purpose is to avoid flickering on animations. If no double-buffering is available, this function is intended to flush any cached drawings onto the window.

void **ge3dHint** (int flag, int value);

This function modifies the ge3d behaviour concerning *hidden surface removal.*
ge3dHint(hint_depthbuffer,0) disables the *z-buffer* and ge3dHint(hint_depthbuffer,1) enables the z-buffer. The function have to be called before activating the drawing mode (ge3d_setmode(int)) to be efficient.
ge3dHint(hint_backfaceculling,1) initiates *backface polygon elimination* and enables *two-sided lighting* (lighting calculations are then correct for the front and back faces of polygons) and ge3dHint(hint_backfaceculling,0) terminates this features (enables one-sided lighting). hint_backfaceculling can be set always. Backface polygon elimination speeds

drawing time by not drawing backfacing polygons (vertices in counterclockwise order).

int **ge3dRequestOverlay** ();

This function and the following functions are used for overlay bitplanes. Overlay bitplanes supply additional bits of information at each pixel. The number of bitplanes (system-dependent) determines the number of colours which can be used in the overlay plane. When zero at a pixel is stored in all overlay bitplanes, the colour of the pixel from the standard colour bitplanes is presented on the screen. If the overlay bitplanes contain a value which is not zero, the overlay value is looked up in a indicated colour table, and that colour is presented instead. Overlay bitplanes are only supported for the GL version and not for OpenGL.
ge3dRequestOverlay should be called once right after initialization to request overlay planes. The return value equals the number of bitplanes which are supported from the system.

void **ge3dBitplanes** (int);

switches between overlay bitplanes and standard (normal) bitplanes.
ge3dBitplanes(ge3d_overlay_planes) activates the overlay bitplanes and all following drawing commands draw in the overlay bitplanes. ge3dBitplanes(ge3d_normal_planes) again activates the standard bitplanes.

void **ge3dClearOverlay** ();

clears the overlay bitplanes (to the invisible color 0) and also activates overlay bitplanes for following commands.

void **ge3dMapColori** (int i, short r, short g, short b);
void **ge3dMapColorRGB** (int i, const colorRGB* RGB);

Overlay bitplanes are always in colour map (colour indexed) mode. Therefore this functions are used in order to choose the colours for drawing in overlay bitplanes. ge3dMapColori and ge3dMapColorRGB define a colour map entry. (r, g, b) are in range 0 to 255, RGB colour components are defined in range 0.0 to 1.0. i is the index for the colour map entry and is in range 1 to (2^n)-1 at which n is the number of overlay bitplanes. Index 0 is reserved for the colour "invisible".

void **ge3dColorIndex** (int i);

activates a colour from the colour map.

D.3 Drawing Modes and Attributes

void **ge3d_setmode** (ge3d_mode_t mode);

sets the drawing mode for shapes, which should be one of:

- ge3d_wireframe
- ge3d_hidden_line
- ge3d_flat_shading
- ge3d_smooth_shading
- ge3d_texturing

void **ge3d_setbackgcolor** (float R, float G, float B); or
void **ge3dBackgroundColor** (const colorRGB*);

Sets the background colour. All *colours* are specified as triples of RGB values in range 0.0 to 1.0.

void **ge3d_setfillcolor** (float R, float G, float B); or
void **ge3dFillColor** (const colorRGB*);

Sets the fill colour (RGB values). Only relevant in modes flat and smooth shading. Also sets the edge colour to (R, G, B).

void **ge3d_setlinecolor** (float R, float G, float B); or
void **ge3dLineColor** (const colorRGB*);

Sets the colour (RGB) for drawing lines and polygon edges. Usually there is a performance penalty for drawing polygons with different line and fill colour; if needed, ge3d_setlinecolor has to be called after ge3d_setfillcolor.

void **ge3d_setlinestyle** (short pattern);

Sets the linestyle pattern, which is specified as a 16-bit integer. For example 0xffff (or -1) is a solid line, 0x0f0f a dashed line.

void **ge3d_setlinewidth** (short width);

Sets the line width in pixels. There may be a performance penalty for drawing lines with widths greater than one or nonsolid lines.

void **ge3dAntialiasing** (ge3d_antialiasing_t flag);

Antialiasing is a method that make objects drawn on a discrete device (the display screen) appear smooth. A line or a polygon drawn without antialiasing have jagged edges because a series of points that are forced to lie on 1 positions on the screen pixel grid build up a line. The flag determines for which objects antialiasing is active: *ge3d_aa_lines*, *ge3d_aa_polygons*, and *ge3d_aa_all* for example ge3dAntialiasing (*ge3d_aa_lines*) activates antialiasing for lines and ge3dAntialiasing (0) deactivates antialiasing for all objects.

int **ge3dAntialiasingSupport** ();

The return value tells the user if antialiasing is supported (TRUE=1) or not (FALSE=0).

D.3.1 The Transformation Matrix Stack

All drawing routines are called with so called *modeling coordinates*, also called object coordinates, because it is the coordinate system in which graphical objects are defined (or modeled). Cameras and light sources are specified in the *world coordinate* system (or scene coordinates). Figure D.1 gives an overview over the coordinate systems from object to window coordinates.

The other coordinate spaces (below world coordinates) are entirely handled by the graph-

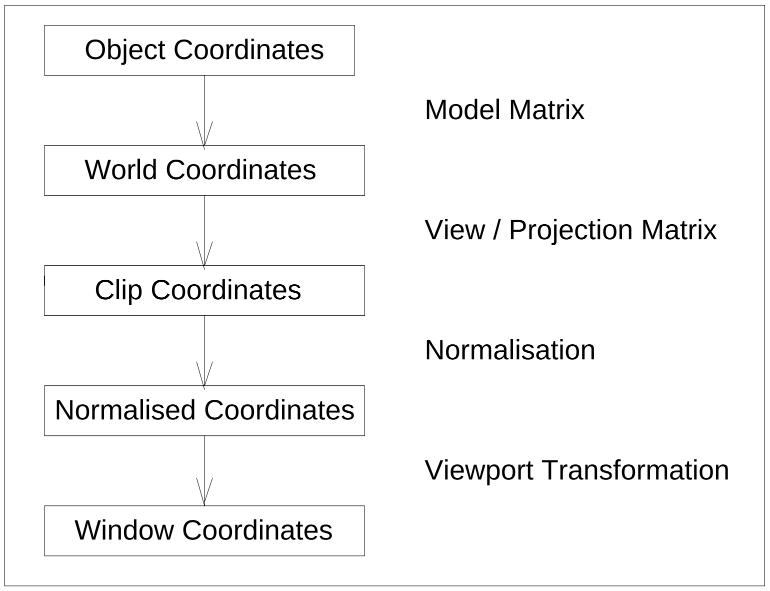


Figure D.1: Coordinate systems

ics library. The camera transformation transforms world coordinates to *clip coordinates* - the view frustum transforms into an axis aligned cube, depth clipping is done in this coordinate system. For z-buffering (hidden surface elimination) the cube is normalised into coordinates in range 0.0 to 1.0 - these are called *normalised coordinates*. At last a simple scaling and translation maps the normalised coordinates to the *window coordinates*.

The transformation from modeling to world coordinates is done with the current *transformation matrix*. All transformation matrices use 4 by 4 homogeneous coordinates and describe *affine* transformations (linear plus translation). Affine transformations include translation, rotation, scaling, and shearing. The matrix need not be built by the user - the GE3D library contains functions to compute it from values for translation, rotation and scaling. (These functions are discussed in the next section.)

Transformation matrices can be stacked on the *transformation matrix stack*. The current transformation matrix is the top matrix on the stack. Other transformation matrices can be pushed onto the stack and later removed. On pushing, the matrix it may be concatenated with the old top matrix, meaning that the transformations are relative to the old one, thus allowing hierarchical description of objects. A typical limit for the maximum depth of the stack set by the underlying graphic library is 64.

There are several routines for handling the transformation stack:

void **ge3d_push_matrix** ();

Pushes down the transformation stack by *copying* the old top matrix. (If the stack was empty an identity matrix is pushed).

void **ge3d_push_this_matrix** (matrix mat);

Pushes down the transformation stack by *pre-concatenating* the matrix mat with the old top matrix. (On an empty stack the matrix is pushed unchanged.)

void **ge3d_push_new_matrix** (matrix mat);

Pushes down the transformation stack and puts the matrix mat *unchanged* onto the top of the stack.

void **ge3d_transform_mc_wc** (float in_x, float in_y, float in_z,
float* o_x, float* o_y, float* o_z); or
void **ge3dTransformMcWc** (const point3D*, point3D*);

Transforms the point (in_x, in_y, in_z), given in modeling coordinates, with the current transformation matrix to the point (*o_x, *o_y, *o_z) in world coordinates. This transformation is applied implicitly to all 3D points when calling any drawing function.

void **ge3d_transformvector_mc_wc** (float in_x, float in_y, float in_z,
float* out_x, float* out_y, float* out_z); or
void **ge3dTransformVectorMcWc** (const vector3D*, vector3D*);

Same as ge3d_transform_mc_wc, but for *vectors*. Note: a translation of a vector makes no sense, therefore the transformation is applied without translation.

void **ge3d_print_cur_matrix** ();

Prints the values of the current transformation matrix to stderr. Can be used as debugging tool. Must not be called on an empty stack.

void **ge3d_get_and_pop_matrix** (matrix mat);

Stores the current transformation matrix in mat and *pops* it from the stack (mat is a reference parameter, because matrix is a float array). Must not be called when the stack is empty.

void **ge3d_pop_matrix** ();

Pops the current transformation matrix from the stack. Must not be called when the matrix stack is empty.

void **ge3dLoadIdentity** ();

Replaces the current transformation matrix with identity matrix.

void **ge3dPushIdentity** ();

Pushes the identity matrix on the stack.

void **ge3dMultMatrix** (const float mat[4][4]);

Premultiplies the current transformation matrix by the given matrix *mat*.

D.3.2 Building Transformation Matrices

The functions *ge3d_translate*, *ge3d_rotate_axis* and *ge3d_scale* build and concatenate transformation matrices for the most common affine transformations: translation, rotation about a coordinate axis and scaling. The computed transformation matrix is *pre-multiplied* to the current transformation matrix.

That means the transformations take effect in the order the functions are called, for example first a translation and then a rotation. Mathematically the transformations have to be applied in the reverse order to get the right result, like first rotating and then translating in the example.

It is an error to call these functions on an empty stack. Function *ge3d_push_matrix ()* should be used first to push an identity matrix onto the stack.

void **ge3d_translate** (float x, float y, float z); or
void **ge3dTranslate** (const vector3D*);

Does a translation by the vector (x, y, z).

void **ge3d_rotate_axis** (char axis, float angle);/*angle in degrees*/

Does a rotation about the axis *axis* ('x', 'y' or 'z') by angle *angle*. The angle is measured in degrees, counter clockwise when looking along the axis towards the origin.

void **ge3dRotate** (const vector3D* axis, float angle);/*angle in radians*/

Does a rotation about a arbitrary axis by angle *angle* in radians (righthand). The rotation axis is given as the vector *axis*.

void **ge3d_scale** (float sx, float sy, float sz, float all); or
void **ge3dScale** (const float* p);

Does scaling with the factor sx along the x axis, sy along y and sz along z, and an overall scaling with the factor all.
The parameter p in the second function is an array of only three values: sx, sy, sz.

D.3.3 Text

void **ge3d_text** (float x, float y, float z, const char* s); or
void **ge3dText** (const point3D*, const char*);

Output of a text string s, beginning at position (x, y, z). The text is written horizontally on the window, beginning at the transformed position. The current line colour is used.

D.3.4 Line Primitives

Some of the functions discussed in this section only have arguments for two dimensions. They are used primarily for 2D drawings, but are also three-dimensional. They draw into the plane z = 0, but are also affected by the current transformation matrix, which can be used to translate and rotate the drawing into the desired position and orientation.

void **ge3d_moveto** (float x, float y, float z); or
void **ge3dMoveTo** (const point3D*);

Moves the current position to the point (x, y, z).

void **ge3d_lineto** (float x, float y, float z); or
void **ge3dLineTo** (const point3D*);

Draws a line from the current 3D position to the point (x, y, z), using the current line colour, style, and width. Then the current 3D position is updated to (x, y, z) for further calls to ge3d_lineto.

void **ge3d_line** (float x1, float y1, float z1,
float x2, float y2, float z2); or
void **ge3dLine** (const point3D* p1, const point3D* p2);

Draws a line from the point (x1, y1, z1) to the point (x2, y2, z2), using the current line colour, style, and width.

void **ge3dPolyLines2D** (float* p);

Draws a arbitrary number of polylines in sequence in the plane z = 0, using the current line attributes. p is an array of float values. Each polyline starts with the number of points, followed by the (x,y) coordinates of the points. The array is terminated with (float) 0. Hence the array contains following sequence of values: n_1 x_{11} y_{11} ... x_{1n_1} y_{1n_1} n_2 x_{21} y_{21} ... x_{2n_2} y_{2n_2} ... n_m x_{m1} y_{m1} ... x_{mn_m} y_{mn_m} 0 (m polylines and n_1 ... n_m points per polyline). The polylines are not automatically closed.

void **ge3d_rect** (float x0, float y0, float x1, float y1);

Draws the outline of an axis-aligned rectangle with opposite points (x0, y0) and (x1, y1) in the plane z = 0. Current line attributes are used.

void **ge3d_wirecube** (float x0, float y0, float z0,
float x1, float y1, float z1); or
void **ge3dWireCube** (const point3D*, const point3D*)

Draws an axis-aligned cube with opposite vertices (x0, y0, z0) and (x1, y1, z1) as wire frame (regardless of the current drawing mode), using the current line attributes.

void **ge3d_circle** (float x, float y, float r);

Draws the outline of a circle with midpoint (x, y, 0) and radius r in the plane z = 0, using the current line attributes.

void **ge3d_arc** (float x, float y, float r,
float startangle, float endangle);

Draws an arc, which is defined as the part of a circle with midpoint (x, y, 0) and radius r (in plane z = 0), beginning at startangle, ending at endangle in counter-clockwise direction. The two angles are given in degrees, measured CCW from the positive x-axis. The current line attributes are used.

void **ge3d_wirepolyhedron** (point3D* vertexlist, vector3D* normallist,
int numfaces, face* facelist);

Draws a wire frame model of a polyhedron (in any drawing mode). The arguments have the same meaning as for *ge3d_polyhedron* (see next section).

D.3.5 Solid Primitives

void **ge3d_circf** (float x, float y, float r);

Draws a circle, filled with the current fill colour (regardless of the current drawing mode). The circle is drawn with midpoint (x, y, 0) and radius r in the plane z = 0.

void **ge3d_rectf** (float x0, float y0, float x1, float y1);
void **ge3dFilledRect** (float, float, float, float, float);

Draws a an axis-aligned rectangle, filled with the current fill colour (regardless of the current drawing mode). The rectangle is drawn with the opposite points (x0,y0) and (x1,y1) in the plane z = 0.
The second function draws a shaded and filled rectangle, and so one can also use this function for *flat shading*.

void **ge3dShadedPolygon** (int n, const point3D* v, const colorRGB* c);

Draws a filled polygon with a colour for each vertex (regardless of current mode). n is the number of vertexes of the polygon and v is an array of 3D vertexes. c is also an array which contains a colour for each vertex. The fillcolour and linecolour have no influence and the mode is switched to gouraud shading.
Filling depends on the current drawing mode (set with ge3d_mode). In mode *wire frame* only the outline of faces is drawn. *Hidden line* does a hidden line elimination which may be achieved by filling the face with the background colour.

Flat and smooth shading take the light sources into account. *Flat shading* uses a single (constant) colour for each face. *Smooth shading* (Gouraud shading) requires normal vectors for the vertices and interpolates the colour smoothly over the face. These two modes also include a hidden surface elimination.

void **ge3d_polygon** (point3D* vertexlist, int nvertices,

```
                    int* vertexindexlist,
                    vector3D* normallist, int nnormals,
                    int* normalindexlist,
                    vector3D* f_normal);
```

Draws a polygon. Parameters:

vertexlist	an array of 3D vertex coordinates (modeling coordinates).
nvertices	the number of vertices of the polygon.
vertexindexlist	an array of nvertices integer indices, telling which vertex of vertexlist is the first vertex of the polygon, the second and so on., in counter clockwise order when seen from outside. The first vertex of vertexlist has index 0.
f_normal	outward normal vector (face normal), used for flat shading. This normal has to be provided for efficiency (to avoid recomputation including normalisation at each drawing).

The other parameters are only used in mode *smooth shading* and if nvertices = nnormals. In this case vertex normals must be provided. If the mode smooth shading is active, but nnormals is less than nvertices, the polygon is flat shaded.

normallist	an array of vertex normal vectors (outward, modeling coordinates).
nnormals	the number of vertex normals of the polygon.
normalindexlist	an array of integer indices, telling which normal vector to use for the first vertex of the polygon, which for the second one and so on. The first normal of normallist has index 0.

That means that the polygon with vertices *vertexlist [vertexindexlist [0]]* to *vertexlist [vertexindexlist [nvertices-1]]* is drawn and automatically closed. In mode smooth shading and if *nvertices = nnormals* the normal vectors *normallist [normalindexlist [0]]* to *normallist [normalindexlist [nnormals-1]]* are used as vertex normals to calculate the colour at the vertices for shading.

For correct results the polygon has to be *convex*. The drawing of a concave or non-simple polygon is undefined but lies within the convex hull. Polygons are *single sided*, therefore the vertices must be given in counter clockwise order when seen from the front.

The number of vertices per face may be limited by the graphics library (e.g. to 255). Normal vectors should be *normalised* to a length of 1.0 for correct shading.

void **ge3d_polyhedron** (point3D* vertexlist, vector3D* normallist,
int numfaces, face* facelist); or
void **ge3dPolyhedron** (point3D* vertexlist, vector3D* normallist,
point2D* texlist , int numfaces, face* facelist);

Draws a polyhedron. The parameters are:

vertexlist	an array of vertex coordinates (modeling coordinates).
normallist	an array of vertex normal vectors (outward, modeling coordinates).
texlist	an array of 2D points for texture.
numfaces	the number of polygons (faces) of the polyhedron.
facelist	an array of numfaces faces.

A *face* contains all the data for one polygon of the polyhedron:

num_faceverts	the number of vertices of a face/polygon.
num_facenormals	the number of vertex normals.
facevert	an array of integer indices into the vertexlist.
facenormal	an array of integer indices to the normallist.
facetexvert	an array of integer indices to the texture vertices.

Note that the vertexlist and normallist (triples of floats) are shared among all polygon faces. Only an integer index is used to specify which vertex or vertex normal is used.

A call of *ge3d_polyhedron* (vertexlist, normallist, numfaces, facelist) leads to the same drawings as a call to *ge3d_polygon* for all faces *facelist [0]* to *facelist [numfaces - 1]* as a loop like

```
int i;
face* faceptr;
for (i = 0, faceptr = facelist;  i < numfaces;  i++, faceptr++)
{
  ge3d_polygon (vertexlist, faceptr->num_faceverts,
                faceptr->facevert,
                normallist, faceptr->num_facenormals,
                faceptr->facenormal,
                &faceptr->normal);
}
```

The use of *ge3d_polyhedron* is in most cases faster because it avoids unnecessary function calls and can draw all faces at once in the proper drawing mode.

Example: A tetrahedron with ground plane in z = 0.

```
static point3D vertexlist [] =          /* vertices */
  {{0, 0, 0}, {1, 0, 0}, {0.5, 0.8, 0}, {0.5, 0.5, 1}};
static int facevert [][3] =             /* vertex indices */
  {{2, 1, 0}, {0, 1, 3}, {1, 2, 3}, {2, 0, 3}};
static vector3D normal [] =             /* face normals */
  {{0, 0, -1}, {0, -0.894, 0.447},
  {0.837, 0.523, 0.157}, {-0.837, 0.523, 0.157}
  };
face facelist [4], *fptr;
int i;

for (i = 0, fptr = facelist;  i < 4;  i++, fptr++)
{ fptr->num_faceverts = 3;
  fptr->facevert = facevert [i];
  fptr->num_facenormals = 0;            /* no vertex normals:*/
  fptr->facenormal = NULL;              /* flat shading */
  fptr->normal = normal [i];
}

ge3d_polyhedron (vertexlist, NULL, 4, facelist);
```

void **ge3dCube** (const point3D* p0, const point3D* p1);

Draws an axis-aligned, shaded and solid cube using the ge3d_polygon function. p0 and p1 are two opposite vertices at which p0.x <= p1.x, p0.y <= p1.y and p0.z <= p1.z.

The following functions support texturing mapping which means that a arbitrary 2-D bitmap is mapped onto a surface in 3-D space for example on a polygon and texturing can be used to enhance visual realism.

int **ge3dTexturingSupport** ();

The return value tells the user if texturing is supported (TRUE=1) or not (FALSE=0).

int **ge3dCreateTexture** (int width, int height,
const void* data,
int bmpformat);

Creates a texture from a data array. The argument *width* and *height* specify the size of the bitmap. *data* is a data array as specified by *bmpformat* argument (currently only one format supported):
ge3d_ubyte_RGB_TB ... triples of unsigned bytes, no filling byte, top-to-bottom
data contains the bitmap of the texture. The function returns a handle which is to be used for later calls of ge3dApplyTexture and finally for destroying with ge3dFreeTexture. In case of an error, 0 will be returned.

void **ge3dFreeTexture** (int handle);

Destroys the texture with the handle *handle*.

void **ge3dDoTexturing** (int toggle);

Switches on (*toggle* = 1) and off (*toogle* = 0) texturing.

void **ge3dApplyTexture** (int handle);

Applies a texture associated with its handle *handle*.

void **ge3dTexturedPolygon** (int nverts, const point3D* vert,
const point2D* tex, int handle);

Draws a polygon where a texture with the handle *handle* is mapped on it. The polygon is textured and flat coloured (regardless of current mode), and fill colour and line colour have no influence. This function contains implicit ge3dDoTexturing (1) and ge3dApplyTexture (*handle*). The polygon has *nverts* vertices which are stored in the array *vert*.
A texture is 2-dimensional and by definition it lies in the range 0 to 1 along two axes in texture space. Now each vertex of the polygon is assigned to a point in the texture space which are stored in the array of 2D points *tex* and so mapping from texture space to geometry in object space is defined. When the (x,y) values in texture space (in the array *tex*) are greater than 1, the texture is repeated.

D.3.6 Camera Definition

GE3D supports a perspective and an orthographic camera model. The perspective camera is appropriate for 3D drawings and the orthographic camera is simpler to specify for 2D drawings. The usage of the procedures is not restricted to these cases, since all drawings are made in 3D space.

The *perspective* camera is defined by a viewpoint position (eye point) and a reference point (lookat) in world coordinates. The distance between eye point and view plane is called focal length and determines together with the aperture (height of the camera window on the view plane) and the aspect ratio (width/height) the field of view.

The orthographic camera also uses a viewpoint position and a reference point for specifying the line of sight, which is projected to the midpoint of the window. The size of the view-port is given by its width and height.

In both cases *depth clipping* is done with two clipping planes called hither and yon. The camera is *untilted* with the y axis as up direction in a right-handed coordinate system.

void **ge3d_setcamera** (point3D pos, point3D ref, float aper,
float focal, float aspect,

float hither, float yon);

Sets up a perspective camera. The arguments have the following meaning:

pos	position of the view point (eye) of the camera (in world coordinates).
ref	reference point (in world coordinates), pos and ref together determine the line of sight, which projects to the midpoint of the view-port.
aper	the height of the camera window on the view plane.
focal	the distance between the viewpoint position and the view plane.
aspect	the aspect ratio of the camera window (width/height, e.g. 4/3).
hither	distance of near clipping plane from viewpoint position, must be > 0.
yon	distance of far clipping plane from viewpoint position, must be > hither.

For a perspective projection *depth clipping* cannot be turned off, because the visible part of the view pyramid, often called view frustum, has to be transformed into a cube. This is also necessary for hidden line elimination with the z-Buffer. If (almost) no depth clipping is wished, set hither to a very low and yon to a large enough number, but setting it too high can cause more rounding errors in the z-Buffer hidden surface algorithm.

The aspect ratio used for setting up the camera should match the aspect ratio of the output window (width divided by height). Otherwise the drawing will be distorted.

void **ge3d_ortho_cam** (point3D pos, point3D ref,
float width, float height,
float hither, float yon);

Sets up an orthographic camera with the following parameters:

pos	position of the view point (eye) of the camera (in world coordinates).
ref	reference point (in world coordinates), the line of sight goes from pos to ref - as in the perspective camera model.
width	the width of the camera view-port.
height	the height of the camera view-port.
hither	distance of near clipping plane from viewpoint position, arbitrary.
yon	distance of far clipping plane from viewpoint position, must be > hither.

For an orthographic camera *depth clipping* is not restricted to regions in front of the eye. It also cannot be turned off, but also drawings behind the eye may be visible.

Example: drawing a diagonal line over the whole window.

```
point3D pos = {0.0, 0.0, 1.0};
point3D ref = {0.0, 0.0, 0.0};
point3D leftbot = {-1.0, -1.0};
```

```
point3D rightup = {1.0, 1.0};

ge3d_ortho_cam (pos, ref, 2.0 /* -1 to 1 */, 2.0 /* -1 to 1 */,
                -10, 10);
ge3d_moveto (leftbot);
ge3d_lineto (rightup);
```

void **ge3dCamera** (int type, const point3D* pos, float rotangle,
const vector3D* rotaxis, float height,
float aspect, float hither, float yon);

One can also use this function to set up a camera using some different and more general parameters. *type* determines the camera model and if the position of the camera is absolute or relative (*cam_perspective*/*cam_orthographic* | *cam_absolute*/*cam_relative*) for example ge3dCamera (*cam_perspective* | *cam_relative*) a relative and perspective camera. A relative camera is placed relative to a object (or the current transformation matrix on the stack). The position of the camera (relative or absolute) is given as *pos* and one can determine the line of sight by rotating the camera about a arbitrary axis by angle *rotangle* in degrees. The rotation axis is given as the vector *rotaxis*. If the camera is a orthographic camera then *height* is the height of the window and if it is a perspective camera then *height* is the total vertical viewing angle in degrees.

D.3.7 Light Sources

For shaded drawings, light sources have to be defined. The library uses *positional* and *infinite* light sources and spotlights in a *diffuse* lighting model. A positional light (a light which is placed at a particular place) is omnidirectional. A infinite light source can be a good approximation for a distant light source and hence one can only determine a light source direction (not a position). A spotlight emits a cone of light that is centered along the spotlight direction.
A diffuse lighting model means that the colour of a surface is independent of the current viewing position and there are no highlights.

Light sources must be first registered with an unique index and can then be turned on and off using that index. Note that the graphic library will limit the number of usable light sources (e.g. only eight in GL).

There are four routines handling the light sources:

void **ge3dSetLightSource** (int index, const colorRGB* RGB,
const point3D* pos, float positional,
int camlgt);

Registers a light source with a (unique) index *index*. The colour of the light is given as *RGB*. If *postional* is zero (infinite light), then *pos* represents a light source direction. The direction must be given towards light source. If *positional* unequals zero, the light source is a positional light at position *pos* (in world coordinates). One can place the light (or set the light direction) relative to the current camera by using the parameter *camlgt* (*camlgt* equals 1).
The index (a small positive integer) is later used to switch on and off the light source. The light is not switched on automatically on registering.

void **ge3dLightSource** (int index, const colorRGB* RGB,
const matrix4D mat, float positional, int camlgt);

This is a more general function to set up a light source. *index*, *RGB*, *positional* and *camlght* have been explained already. Position/orientation is not given as a point/vector, but a transformation matrix that transforms the default position (0, 0, 0) or orientation (-1, 0, 0).

void **ge3dSpotLight** (int index, const colorRGB*,
const point3D* pos, const vector3D* dir,
float droprate, float cutangle);

Registers a spotlight with the index *index*, which is placed at position *pos* (in object coordinates) with the spot direction *dir*. A spread angle *cutangle* of for example 45 defines a cone with a radius angle of 45 degrees (range 0 to 90 or 180). The intensity falloff is given as *droprate*.

void **ge3d_switchlight** (int index, int state);

Switches the light source with index index on (if state is not 0) or off (if state is 0).

D.3.8 Closing the Graphics Device

void **ge3d_close** ();

Closes the graphics device.

Bibliography

[1] Keith Andrews, Frank Kappe, and Hermann Maurer. Hyper-G: Towards the next generation of network information technology. *Journal of Universal Computer Science*, 1(4):206–220, April 1995. Special Issue: Selected Proceedings of the Workshop on Distributed Multimedia Systems, Graz, Austria, Nov. 1994. Available at `http://-hyperg.iicm.tu-graz.ac.at/jucs_root`.

[2] Keith Andrews, Frank Kappe, and Hermann Maurer. Serving information to the web with hyper-g. *Computer Networks and ISDN Systems*, 27(6):919–926, April 1995. Proc. 3^{rd} International World-Wide Web Conference. Available at URL `http://www.elsevier.nl/www3/welcome.html`.

[3] Keith Andrews, Frank Kappe, Hermann Maurer, and Klaus Schmaranz. On second generation network hypermedia systems. In *Proc. of ED-MEDIA 95*, pages 69–74, Graz, Austria, June 1995. AACE.

[4] Gavin Bell, Anthony Parisi, and Mark Pesce. The virtual reality modeling language – version 1.0 specification. Available at `http://vrml.wired.com/vrml.tech/vrml10-3.html`, May 1995.

[5] Emily Berk and Joseph Devlin, editors. *Hypertext/Hypermedia Handbook*. Software Engineering Series. McGraw-Hill, New York, 1991.

[6] Mark Bernstein. The navigation problem reconsidered. In Emily Berk and Joseph Devlin, editors, *Hypertext/Hypermedia Handbook*, Software Engineering Series, pages 285–297. McGraw-Hill, New York, 1991.

[7] Ron Britwich. Web World. Available at `http://sailfish.peregrine.com/WebWorld/welcome.html`.

[8] Vannevar Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, July 1945.

[9] Matthew Chalmers. Using a landscape metaphor to represent a corpus of documents. In Andrew U. Frank and Irene Campari, editors, *Spatial Information Theory A Theoretical Basis for GIS*, pages 377–390. Springer-Verlag, 1993.

[10] Kim Michael Fairchild. Information management using virtual reality-based visualizations. In Alan Wexelblat, editor, *Virtual Reality: Applications and Explorations*, pages 45–74. Academic Press, 1993.

[11] Kim Michael Fairchild, Steven E. Poltrock, and George W. Furnas. SemNet: Three-dimensional representations of large knowledge bases. In Raymonde Guindon, editor, *Cognitive Science and its Applications for Human-Computer Interaction*, pages 201–233. Lawrence Erlbaum, Hillsdale, NJ, 1988.

[12] George W. Furnas. Generalized fisheye views. In *Proc. CHI'86*, pages 16–23, Boston, MA, April 1986. ACM.

[13] Geri Gay and Joan Mazur. Navigating in hypermedia. In Emily Berk and Joseph Devlin, editors, *Hypertext/Hypermedia Handbook*, Software Engineering Series. McGraw-Hill, New York, 1991.

[14] M. Henke. Hypermedia in der medizin - die gestaltung der digitalen patientenakte als hypermedium. In H. P. Frei and P. Schäuble, editors, *Hypermedia '93, Zurich, Switzerland*, pages 172–182, Berlin, mar 1993. Springer.

[15] Wolfgang Dalitz / Gernot Heyer. *Hyper-G Das Internet-Informationssystem der 2. Generation.* dpunkt Verlag fuer digitale Technologie GmbH, Heidelberg, Oktober 1995.

[16] Ed Krol. *The Whole Internet: User's Guide and Catalog.* O'Reilly & Associates, second edition, April 1994.

[17] Mark A. Linton, John M. Vlissides, and Paul R. Calder. Composing user interfaces with InterViews. *IEEE Computer*, 22(2):8–22, February 1989.

[18] Alexander Willett Matthias Hemmje, Clemens Kunkel. Lyberworld - a visualization user interface supporting fulltext retrieval. In W. Bruce Croft and C. J. van Rijsbergen, editors, *Proc. SIGIR '94*. Springer-Verlag, July 1994.

[19] Hermann Maurer. *Hyper-G: The Next Generation Web Solution.* Addision-Wesley, 1996.

[20] Mark P. McCahill and Thomas Erickson. Design for a 3D spatial user interface for Internet Gopher. In *Proc. of ED-MEDIA 95*, pages 39–44, Graz, Austria, June 1995. AACE.

[21] Jakob Nielsen. *Multimedia and Hypertext: The Internet and Beyond.* Academic Press, San Diego, CA, 1995.

[22] A. M. Pejtersen. Designing hypermedia representations from work domain properties. In H. P. Frei and P. Schäuble, editors, *Hypermedia '93, Zurich, Switzerland*, pages 1–29, Berlin, mar 1993. Springer.

[23] Mark Pesce. *VRML: Browsing and Building Cyberspace.* New Riders/Macmillan, 1995.

[24] Michael Pichler. Interactive browsing of 3D scenes in hypermedia: The Hyper-G 3D viewer. Master's thesis, Graz University of Technology, Austria, October 1993.

[25] Michael Pichler, Gerbert Orasche, Keith Andrews, Ed Grossman, and Mark McCahill. VRweb: A multi-system VRML viewer. In *To appear in Proc. First Annual Symposium on the Virtual Reality Modeling Language (VRML 95)*, San Diego, California, December 1995.

[26] George G. Robertson, Stuart K. Card, and Jock D. Mackinlay. Information visualization using 3D interactive animation. *Communications of the ACM*, 36(4):56–71, April 1993.

[27] Luis Serra, Tat-Seng Chua, and Wei-Shoong Teh. A model for integrating multimedia information around 3d graphics hierarchies. *The Visual Computer*, 7(5-6):326–343, May/June 1991.

[28] Simon Shum. Real and virtual spaces: Mapping from spatial cognition to hypertext. *Hypermedia*, 2(2):133–158, 1990.

[29] Pauline A. Smith and John R. Wilson. Navigating in hypertext through virtual environments. *Applied Ergonomics*, 24(4):271–278, August 1993. Butterworth-Heinemann Ltd.

[30] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley, Reading, Massachusetts, second edition, 1991.

[31] Joel Tesler and Steve Strasnick. *FSN: The 3D File System Navigator.* Silicon Graphics, Inc., Mountain View, CA, 1992. Available by anonymous ftp from `sgi.sgi.com` in directory `sgi/fsn`.

[32] John A. Waterworth and Gurminder Singh. Information islands: Private views of public places. In *Proc. of East-West International Conference on Multimedia, Hypermedia, and Virtual Reality (MHVR'94)*, pages 201–206, Moscow, Russia, sep 1994.