Dynamic Potential Fields for Guided Exploration in Virtual Environments

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieurin (Dr.-Ing.)

angenommen von der Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von	Dipl.-Phys.Ing. Steffi Beckhaus, MSc
geb. am	14.12.1967 in Essen
Gutachter:	Prof. Dr. Thomas Strothotte
	Prof. Dr. Heinrich Müller
	Dr. Vali Lalioti

Ort und Datum des Promotionskolloquiums:
Magdeburg, 13. September 2002

Acknowledgements

This work would not have been possible without the support of other people. Prof. Dr. Thomas Strothotte had a lot of confidence in me and encouraged me throughout the past three and a half years to work on this topic. Prof. Dr. Heinrich Müller always was a source of support and ideas with a focus on essentials and detail. Dr. Vali Lalioti greatly helped me with advise on virtual environment questions. Andreas Simon spent numerous hours on reading and discussing this thesis and helped a lot in structuring this work. Jürgen Wind was an endless source of information and support when it got to implementation issues. Felix Ritter and Dr. Gerhard Eckel supplied their applications, and Jeremy Eccles built the model of the Bonn Marktplatz application. Aeldrik Pander helped in making the videos, and Melinda Cerney and Kris Blom did the tedious job of proof-reading. I would also like to mention Jürgen Landers and all my family for giving their support and being patient with me during the time that was needed to finish this work. There are many more people, from the Virtual Environment (VE) group at Fraunhofer IMK, from outside the group, and many friends who contributed with information, inspiration, and encouragement. I truly wish to thank all of these people for their invaluable support.

Zusammenfassung

Exploration in räumlichen Umgebungen ist ein dynamischer, interaktiver Prozess, der aus einer kognitiven Komponente, u.a. zur Identifikation von möglichen Zielen, und einer Bewegungskomponente, nämlich der Bewegung von der aktuellen Position zu einem identifizierten Ziel, besteht. Wir schlagen ein System vor, das diese beiden Komponenten von Exploration in virtuellen (VR) Umgebungen unterstützt. Zur Auswahl der räumlichen Ziele werden die Ergebnisse von Abfragen an ein Informationssystem in unsere Methode integriert. In dem resultierenden Zielfeld, bestehend aus einem oder mehreren Zielen, die eine unterschiedliche Relevanz zur Abfrage haben können, bewegt sich die Kamera automatisch zu den einzelnen Zielen und präsentiert sie somit.

Unsere Methode benutzt dynamische Potentialfelder, eine lokale Methode, die von der Physik einer Ladung in einem elektrischen Feld abgeleitet und weiterentwickelt worden ist. Zur Generierung der von dieser Methode benötigten diskreten Repräsentation der geometrischen Umgebung, stellen wir zwei echtzeitfähige Voxelisierungsmethoden vor. Das hier präsentierte System, das CubicalPath System, kann interaktive Eingaben und Veränderungen von Parametern durch Applikation und Benutzer verarbeiten. Diese können sein: bewegliche, nicht vorhersagbare, autonome Objekte, bewegliche und neu definierte Ziele, veränderte Kameradaten der Anwendung und Eingaben durch ein Interaktionsgerät. Das System ist plattform-unabhängig und konzipiert als optionales Unterstützungssystem, das wie ein zusätzliches Interaktionswerkzeug in die Applikation eingebunden werden kann. Die Datenbank und die Anwendung kommunizieren mit dem Serversystem über schlanke, einfache Schnittstellen, welche die allgemein benutzten Datenstrukturen widerspiegeln.

Wir haben drei existierende Anwendungen erfolgreich um unseren Ansatz zur unterstützten Exploration erweitert. Mit einem informellen Benutzertest wurden verschiedene Aspekte des Ansatzes und seiner Benutzbarkeit validiert. Mit dem CubicalPath System lassen sich einfach externe Anwendungen um ein ziel-orientierte Abfragen ermöglichendes Informationssystem erweitern und der Benutzer wird davon befreit, direkt, manuell in einer virtuellen Umgebung steuern zu müssen. Dadurch wird es ihm ermöglicht, sich besser auf die dargestellten Inhalte konzentrieren zu können und die räumlichen Inhalte besser aufnehmen zu können.

Abstract

Exploration is a highly interactive, goal-driven process which involves decision making and travelling for the purpose of discovery. We propose a system for guided exploration in a virtual environment, a system which supports the user while exploring the environment. We support target selection by integrating the output of queriable information spaces into our system. We support travelling to these targets by automatically moving the user inside the resulting goal field, presenting the targets one-by-one.

Our approach is based on the concept of dynamic potential fields, a local method derived from the physics of the motion of a charged particle – here a camera – in an electric potential field. It uses a discretized representation of the environment in a uniform rectangular grid. For the geometric setup, we present two, real-time capable voxelization methods. We present the CubicalPath system which is able to deal with interactive, real-time input by both the client application and the user, which can consist of dynamic, unpredictable object locations, dynamic or re-adjusted targets, a modified application view, and a force input generated by an interaction tool. The system is designed to be a platform and machine independent auxiliary support system and can be used by the application as an additional interaction tool. The application and the information space are connected to the server system through a lean interface which mirrors their common data structures.

We applied our system successfully to three existing applications and validated it with an informal usability test. The results show that the system is easily integrated into existing virtual environment applications and was generally found to be a useful support for exploration. With the CubicalPath system, existing applications can easily be enriched with a additional information layer, queriable by the users, and it relieves users from the task of directly, manually navigating in a virtual environment. This enables them to concentrate on the spatial information and to learn more about the presented environment.

Contents

1 Introduction

1.1 Motivation

Exploration of unknown territory is a highly interactive task in which the person who wishes to explore a scene browses through the environment while discovering and gathering information. In this process, users frequently adjust their current personal focus – the goal that drives the movement and the interaction – because of the discoveries made and the knowledge gained. This is the cognitive part of exploration. The initiation and control of the actual movement to the identified spatial target is the motion or travel part of exploration. Both parts are tightly connected [Con01].

In virtual environments (VE), exploration is an even more complex process than in the real world [Bow02]. Easy and intuitive real world tasks, like walking and looking at things or touching objects and picking them up, are much more demanding in virtual worlds. All behavior has to be simulated and controlled through an interface. Effective, intuitive control is rarely achieved in virtual environments. In virtual environments, natural and trained behavior for motion and interaction, which is normally done unconsciously, cannot be used one-to-one. An interaction tool has to be utilized to provide this interface between the user and the system. With the current technical possibilities, tactile feedback can, if at all, only be given by means of a substitute. This makes interacting with the virtual world much more tedious than interacting in the real world. The process of interaction takes away some of the user's attention, which normally would be fully focused on the exploration task.

We think that there is a possibility to support explorative tasks in virtual environment in two ways. First, there can be support on the cognitive level. It is common practice to people to query the internet via search engines in order to explore a question. Why not provide for the possibility to query application information spaces in order to identify targets that meet the goal? Second, there can be support for the navigation task in the environment. If the navigation task requires too much focus, a process of discovery hardly takes place. Coupling information spaces with navigation support would result in a system which supports exploration in an extensive way. We could enrich dull geometric 3D scenes in a virtual environment with contextual information without modifying the environment or requiring good navigation skills. Or we could simply provide for a navigation aid to targets.

1.1.1 Guided Exploration

There are several reasons for exploration in virtual environments. An exploration task might be needed to gain knowledge about the (spatial) composition of a complex system, or it can be required during a simulation task, such as the assembly of parts of a composite object [RPDS00]. These concepts of problem solving and doing simulation play an important role in teaching and learning theories. As stated by the cognitive psychologist Bruner [Bru76], for effective learning, the independent acquisition of knowledge and the solving of problems is most important. Learners should solve given tasks by themselves. By doing so, they inquire into the subject and also practise the necessary techniques. Different from other learning strategies which rely on confronting the students with pre-structured, abstract knowledge, here the students abstract knowledge themselves. They learn by *exploring* the subject on their own, which also teaches them independence and self-learning abilities. Should problems arise, they may be *guided* by a teacher. This leads to the definition of the term *guided exploration*:

Guided Exploration is the independent acquisition of knowledge by exploring the available information. Guidance is provided as a reaction to user-initiated questions in the form of hints or results of the queries. These results are then presented in an appropriate way.

In virtual environments, the available information consists of geometric and, possibly, of contextual information. The answers to user initiated questions are spatially distributed objects which can be presented by moving the user to these objects.

1.1.2 Information Space

Guided Exploration requires a system which provides for contextual information. We rely on the existence of an underlying information space which holds information about the objects in the scene. Strothotte [Str98] defines information space as the combined information about the geometric and the symbolic model. The geometric model defines the physical appearance of objects, while the symbolic model defines all other data available for the object.

The user may query the information space in order to explore the symbolic information. A query results in a list of objects that match the query criteria, and may be a simple search for objects that match a specified attribute. The smallest possible version for the information space would be a list of objects which is made available to the user for selection. If the information space also incorporates semantic context and complex information dependencies [Har01], then the query can be arbitrarily complex. The result of such a query will be a list of objects which may be qualified with a value that denotes each object's relevancy to the query. This value ranks objects as more or less important in the context of the query. Because the objects are distributed in the geometric model, the result of a query forms a three-dimensional field of spatially distributed relevancy values. This field is called the *goal field*, as the query was driven by a user-defined goal. This relationship is illustrated in Figure 1.1.

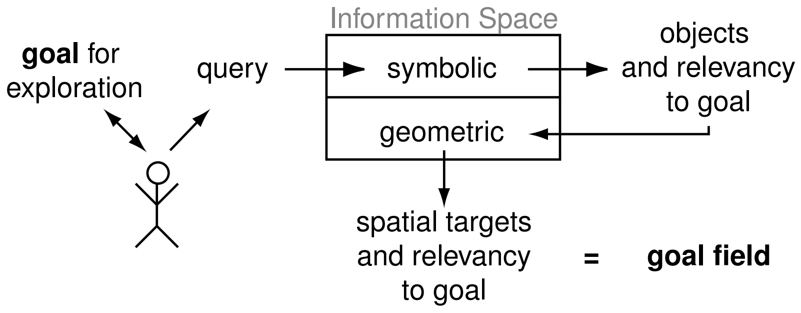


Figure 1.1: From Goal to Goal Field.

1.1.3 Virtual Environment Applications

We aim at developing techniques for applications which use immersive projection-based display systems for presenting stereoscopic images to the user. Nevertheless, the approach can also be used in desktop applications. In virtual environments, it is difficult to incorporate contextual information into the 3D world. Additional windows with textual or other information, common in desktop computer applications, are not appropriate, as they would largely disturb the immersive feeling and would occlude objects in the 3D scene. Text information is not suitable because of its 2D nature. A one-by-one view of the results without spatial transition (slide show) or a concurrent view of the results taken out of their context (image overview) is not desirable in large-scale virtual environments, as these methods would again detach the user from the immersive experience and could cause spatial disorientation. Thus, for large-scale virtual environments, other methods for incorporating information into the scene are necessary.

Guided exploration provides for a method of transferring information (specified by the user through a question or a query) by presenting the results (which are objects in a spatial context) to the user. The presentation of the objects is done by moving the eye-point of the user through the environment and bringing all objects of interest one after the other into the view. In this way, undesirable jumps or cuts are avoided.

1.2 Goal

In this thesis, the goal is to develop a method which can support the user in the highly interactive task of exploring a 3D virtual environment, thus a method to provide guided exploration. We want to integrate the results of queries to existing application information spaces into the virtual environment. The presentation of these results should not change the environment by adding or modifying the objects, or by separating objects from their spatial context. Therefore, we want to visualize the results by moving the user to the identified spatial targets. In this way, we support both the cognitive element of exploration and the core travel task.

A major emphasis is on interactivity. The supportive system should leave a great deal of freedom to the user in order to prevent suppressing the user's spontaneity. The user should be able to determine at each moment if they directly control interaction with the system or if the system supports them in their task. The system should be designed in such a way that it is easily integrated into existing applications. Ideally, it produces output which can replace the output generated by an interaction tool, such that the two can easily be interchanged[1]. In addition, the system should be able to immediately react to new targets and dynamic environments. This requires a real-time method which completely avoids planning of a path, as planned routes may be discarded due to user changes to the current goal and possible modifications in the spatial arrangement of objects. Planning long routes also does not allow for real-time reaction to user queries and other user interaction.

1.3 Results

The major results of this thesis are the following:

- **Automated Travel**
  In this thesis, we develop a method to automatically move the user of a dynamic, virtual environment application to a specified target location. The method is based on potential fields, but uses force functions to speed up computation. It separates the geometric and the target setup. Both layers can be modified at any time during execution to depict the current state of a dynamic application like moving obstacles or targets. We introduce the first system-independent method for virtual environments which automatically moves the camera to a static or dynamic target in dynamic, unpredictable environments and provides for collision prevention with objects. It is controlled through a lean interface, mirroring common application data structures.

[1]The interaction tool is normally connected to the viewer matrix of the virtual environment system by a transformation matrix, thus directly manipulating the view. By using this transformation matrix, external calculated motion through the virtual environment can easily be integrated.

- **Guided Exploration**
  We look into a method which provides guided exploration in virtual environments. Guidance is introduced by moving the users to targets of interest, which are selected by queries to existing application information spaces. The resulting goal field, a distribution of targets with a weighted relevancy, their attractivity to the camera, is then travelled by the supportive system to present the targets, one-by-one to the user.

  We develop the method of dynamic potential fields, which deals with multiple targets by continuously decreasing the attractivity of visible targets. Basic support for specific views on objects is incorporated by means of separating positional and orientational targets and by including navigation objects– additional, often invisible, targets or repulsive objects. During run-time, the system allows for interactive input of all parameters to the system: dynamic object locations, dynamic or re-adjusted targets, a change of the application camera, or the additional influence of a force input generated by a client interaction tool. With navigation objects, we also can prevent the camera from moving into unwanted areas or dead ends. The implementation of this method results in the first system which supports exploration in a dynamic, unpredictable virtual environment by incorporating results of queries to an application information space into a motion generation system which then travels to all specified targets.

- **Real-time Voxelization**
  To provide for methods for fast discretization of dynamic environments, required by potential field-based methods, we develop two methods for real-time voxelization. The software-based method is a 3D scan conversion based triangle voxelizer, which transforms each object's polygons into the voxel space. The hardware-based method renders and analyzes the complete voxel space to retrieve the objects inside each voxel. A special approach to color-coding allows the identification of multiple objects per voxel, to make the results comparable to the results of the software-based method. Both methods show their strength for different types of object complexity and environment setup.

- **Autonomous, Auxiliary Support System**
  Our method for automated travel and exploration support is implemented in the system-independent client/server based CubicalPath system. This is controlled through a lean interface and communicates with the application via CORBA. It makes use of common data structures available in typical virtual environments, like the scene-graph and transformation matrices, connecting the viewer and an interaction tool. The CubicalPath system can easily be integrated into existing virtual environment applications in exchange or in addition to any other typically available interaction tool like, for example, a joystick.

- **System Evaluation and Usability Testing**
  We successfully prove our concept and system with three applications. One

is a desktop application, and the other two make use of immersive virtual environments. We evaluate the system in an informal user study. The results support our judgement that users welcome guided exploration, and that automated travel is not felt to be distracting or disturbing. Furthermore, our study shows that current interfaces to system control in virtual environments, required for querying databases, are tedious to handle. As they are important as an interface to complex information spaces, they strongly deserve further research.

1.4 Overview

Chapter 2 explains about virtual environments and their applications in general. It then describes interaction in virtual environments with a specific focus on one of the major tasks in virtual environments and the core task for exploration: travel, the navigation from a current position to a target location. Known travel metaphors are listed. After that, the more general task of exploration and supportive approaches to exploration are investigated. The term goal field is coined, denoting a field of multiple targets generated from the results from a goal-directed query to an application information space.

It was found that common travel metaphors always rely on direct user control. Simple support for motion to a target relies on the selection of visible targets or selection from a list, and results in straight motion without collision control. There is no system available which provides automated travel in virtual environments in an application-independent way. The only system which supports exploration, does this by constraining the motion of a user to a predefined constraint field. It cannot cope with dynamic environments, does not include automated motion, and has no notion of travelling a goal field.

Chapter 3 introduces a new automated travel method to relieve users from the tedious task of travelling to a known or spatially unknown target. The method is based on the behavior of a charged particle in an electric potential field, where the target attracts the camera, while obstacles repulse it. It is a local, real-time capable method which uses a discretized environment, the cube space, as a uniform data structure.

We introduce the first system-independent method for virtual environments, which automatically moves the camera to a static or dynamic target in dynamic, unpredictable environments and provides for collision prevention with objects. Our implementation provides for a system-independent, automated travel system which is controlled through a lean interface by common, virtual environment system data structures, like the scene-graph or transformation matrices, connected to the application viewer. One of the drawbacks of the local method, the stop of the camera in an unwanted local minimum, can arise in cluttered and maze-like environments, but is rare in sparse environments. Exploration requires the presentation of multiple targets, positions in space, which is not possible with this method as the camera will stop moving once it has arrived at the first target.

Chapter 4 extends the automated travel system to be able to move in a goal field, a field of multiple targets qualified with a level of relevancy to the user in the context of the goal. This field is mapped to the attractivity field of the potential field, thus mapping relevancy to attractivity of targets. We introduce dynamic potential fields, which deal with multiple targets by continuously decreasing the attractivity of visible targets. This, in effect, releases the camera from visited targets after a period of time. Basic support for specific views on objects is incorporated by means of separating positional and orientational targets, and by including navigation objects– additional, often invisible targets or repulsive objects.

The implementation of this method results in a system which provides guided exploration in a dynamic, unpredictable virtual environment by incorporating results of queries to an application information space into a motion generation system which then travels to all specified targets. The targets are presented one-by-one, the order based upon their level of attraction to and their distance from the current camera position. During run-time, the system allows for interactive input of all parameters to the system: dynamic object locations, dynamic or re-adjusted targets, a change of the application side camera, or the additional influence of a force input generated by a client side interaction tool, which is added to the results of the motion generation function. With the use of navigation objects, we are able to view targets from a specific side or angle, and we can prevent the camera from moving into unwanted areas like a dead end.

Chapter 5 introduces two real-time capable voxelization methods, which are used for the discretization of the environment. This is required for the geometric setup of the potential field methods, especially if the environment is dynamic and frequent re-voxelization is necessary. Both methods generate object-coded volume data (each volume element is assigned a list of occupying objects) from geometric surface data by utilizing the Performer$^{\text{TM}}$scene-graph. As the inside of objects is not visible, it makes no sense to include these data into the calculation. Thus, surface data is used. In the software-based method, a 3D scan conversion based triangle voxelizer, each object's polygons are transformed into the voxel space. The hardware-based method renders and analyzes the complete voxel space to retrieve the objects inside each voxel. A special approach to color-coding allows the identification of multiple objects per voxel, making the results comparable to those of the software-based method.

The performance measurements show that the software-based method is preferred for its simplicity and its real-time behavior if only a small number of objects are dynamic, while the hardware-based approach shows its strength if the *z resolution* – the height of the space – is small compared to the *x* and *y resolution* – the large area that can be navigated. Both methods in their specific range are sufficiently fast to support the applications we use later. For very large resolutions, the discretization part of the algorithm will be the bottleneck of the potential field method in dynamic environments.

Chapter 6 describes the implementation of the CubicalPath system to give an overview of the interfaces to the system and the requirements and amount of work involved to utilize it both from a programmer's and a user's view. The interface implementation is described in an example client application which is based on AVANGO, the Fraunhofer Institute for Media Communication's (IMK) framework to virtual environment application development.

Using CORBA, the CubicalPath system is a system-independent client/server system, which works as an auxiliary support system. The virtual environment application and the application information space are connected to the server system through a lean interface, which mirrors their common data structures. These data structures are the scene-graph, the transformation matrix connected to the application viewer, and a list of object data resulting from a query to an information space. Continuously generated camera data is output from the system to the client application and then connected to the transformation matrix of the client application viewer. The lean interface makes the CubicalPath system easily integrated into existing virtual environment applications.

Chapter 7 presents three existing applications which are extended by connecting them and a database to the CubicalPath system to use the system's features for providing guided exploration. The applications are a medical training system run on a desktop computer, a virtual art museum run in a CAVE^{TM}-based system, and the Bonn Marktplatz application run in the i-Cone^{TM}, developed by Fraunhofer IMK.

The integration of the system into the applications proved to be straightforward even though different systems and scene-graph types were used. The short setup time showed the usability of our interface concept. Nevertheless, it was found that even though an initial motion of the camera to targets could quickly be triggered, some time has to be devoted to adjusting motion-specific and system control parameters to achieve a pleasant task and application-specific motion.

Chapter 8 presents an informal user study carried out with ten users to obtain an initial evaluation of our approach to guided exploration both on a technical and a cognitive level. We designed a questionnaire to collect information about personal feeling, assessment of features, and content learnt.

The result of our study showed that the supportive system was well received by the users without depriving them of the feeling of presence and self-control. This indicates that it is possible to use self-navigation, automated travel, and guided exploration in a virtual environment side-by-side without confusing the users. The concept of guided exploration within the system was rated as very helpful by most users and helped in structuring information for classification and ease of recall. It was also found that the users could concentrate much more on the task itself if they were relieved of the travel task. The study showed that considerable research is necessary to improve interfaces to system control in virtual environments.

Chapter 9 summarizes and concludes the content and results of this thesis and gives some ideas for future work.

2 Foundations

The aim of this thesis is to develop a system which is able to provide support for exploration, termed guided exploration, in virtual environments. In this chapter, we look into virtual environments, travel, exploration, and motion generation techniques. We start by explaining virtual environments in general and their applications, systems, and interaction techniques and metaphors. Because navigation, travel and exploration are closely related and often confused terms, we will explain these and define their meaning in our context. For travel and exploration, we will look into supportive metaphors or systems. As we want to support exploration by automatically moving through the environment, we will then look into computer-generated motion techniques which are suitable for transfer to virtual environments. Here, we will consider both pure path planning approaches as well as motion generated for the purpose of presentation.

2.1 Virtual Environments

Virtual environments (VE) can be defined as real-time, 3D, computer-generated environments which create some feeling of presence of the user in the environment, the "sense of being there". Virtual environments can be composed of visual, auditory, tactile, olfactory and vestibular information that is rendered and displayed with appropriate systems and is controlled with various tools that depend on the system and the application.

For stereo viewing, the visual part of a virtual environment (e.g. the geometry) is rendered independently for each eye's perspective so that the user gets a 3D impression of the scene (stereo visualization). With the help of a tracking system, the user's position and viewing direction is followed by the computer and the perspective is updated correspondingly. In most cases, the user can interactively move through the virtual environment and examine the data or model.

2.1.1 Historic Overview and Applications

The pioneers of virtual environments were Heilig in 1956, who showed 3D movies with his multisensory Sensorama, and Sutherland in 1963 who provided for the first, interactive computer graphics with his Sketchpad [Rhe91, Sut63]. Sutherland also developed the first head-mounted display (HMD). Since the sixties, major advances in the visual rendering have been made, and in 1992, the first immersive surround-screen projection-based system, the CAVE^{TM}, was developed by Cruz-Neira [CNSD93]. Advancements in projection-based systems include the Responsive Workbench^{TM}in 1993 [KF94] and the i-Cone^{TM}in 2001, both developments of the Institute for Media Communication at Forschungszentrum Informationstechnik (GMD) in Sankt Augustin, which is now the Fraunhofer IMK.

Today, a large number of application domains can benefit from virtual environments, namely architecture, education, manufacturing, medicine, simulation & training, entertainment, design & prototyping, information & scientific visualization, and collaboration & communication. These applications may draw from the fact that users feel present in the environment and can act naturally. Such application domains utilize the virtual environments to present and interact with 3D spatial information, models and scenes in 3D, to create a true spatial experience.

2.1.2 Presence and Immersion

One of the main features of virtual environments is that they can produce a sense of presence in the user, a true feeling of "I am part of the virtual world". One example of an evaluated system which produces this feeling is Disney's Aladdin, where guests fly a magic carpet through a virtual world based on the animated film "Aladdin" [PST^{+}96]. Witmer and Singer define presence as the "subjective experience of being in one place or environment, even if one is physically situated in another" [WS94]. They list factors, grouped as control, sensory, distraction and realism, which influence the sense of presence. They define immersion as the person's response to the virtual environment system. We prefer the definition of Slater and Wilbur [SW97] or Regenbrecht and Schubert [RS97], who define immersion as the hardware- and software components – like appropriate resolutions, field-of-view, and invisibility of borders – that are needed to bring the actor inside a virtual world. To create immersion, distractive influences like the boundaries or shape of the display system, unsuitable interaction tools or metaphors, or obstacles occluding the view have to be minimized. For example, a large field-of-view and stereo viewing may enhance the sense of immersion in the virtual world. Intuitive, near-reality interaction methods are an important factor for the level of immersion in a virtual environment. These are some of the factors for immersion which are, in effect, the pre-requisites to generating a sense of presence. Presence, then, defines the cognitive process of constructing an environment which, as a result, leads to the user's perception of being part of it.

Two important areas of virtual environment research, *Virtual Environment Systems* and *(Human Computer) Interaction,* are concerned with improving the virtual environment experience. We will give an overview of relevant research in these areas in the following.

2.1.3 Virtual Environment Systems

Systems which generate virtual environments differ in their display setup, viewing techniques (mono or stereo, active or passive stereo), usable interaction metaphors and techniques, and the possible level of immersion achieved. While mono systems are mostly non-immersive – they show a virtual environment in 2D on a screen – stereo systems and large field-of-view systems can be semi-immersive or fully-immersive. Examples for semi-immersive stereo systems are the Responsive Workbench and the i-Cone, while the six-sided CAVE and the head-mounted displays (HMD) are considered to be fully-immersive. A stereoscopic HMD has two displays, one for the left and one for the right eye. The HMD is fixed to the user's head, and each eye can only see the associated display. The users can see themselves and tools only with the help of avatars and virtual objects. The Responsive Workbench, the CAVE, and the i-Cone system are *projection-based* systems. These all run in active-stereo mode, which means that the images for each eye are rendered alternatively onto the same screen and are separated for the eyes with shutter glasses (e.g. Crystal Eyes^{TM}). Users are detached from the display, thus can see themselves, other users and the interaction tools used. The projection-based systems Responsive Workbench, CAVE, and i-Cone are described in Appendix E.

A distinctive feature of stereoscopic systems is *head-tracking.* A 3D object is experienced by humans through subtle movements of the head, thus building a 3D representation of this object in the brain. This natural method of object exploration can be integrated into the virtual environments by using head-tracking. By tracking the position of the head, the exact position of the eye is known, and can then be used to generate the precise view of the user. This allows for an intuitive perception of the 3D world[1].

Large field-of-view systems like the i-Cone still give a strong immersive and 3D feeling even though they do not employ head-tracking. This is because the user is surrounded by the display, therefore inside the world and the screen, thus the objects are so distant that small head movements would not lead to a noticeable change in the stereo image.

[1]Head-tracking is, in most cases, only enabled for one person per display, as otherwise $user * 2$ (one for the right eye, one for the left eye) views have to be rendered and displayed in the same time. In multi-user environments, this means that non-head-tracked users have to keep close to the tracked person to have the correct, undistorted stereo view.

2.1.4 Interaction

In addition to allowing for a true spatial experience (for example a large field of view and stereoscopic rendering), a virtual environment system must have some means of interacting with the virtual environment. Virtual environments require completely different interaction techniques than do desktop computers. For example, there is no desk on which to move a mouse, and there is a strong need for 3D spatial interaction. Usually, the user has some interaction tools so that he or she can accomplish tasks such as controlling the system, navigating, selecting and modifying objects. These tools consist of hardware components – the input device – and software components, like the mapping of hardware sensor values to virtual world behavior. The choice of tools depends on the available input devices and on the task that should be accomplished by the user. Interaction can strongly influence the user's sense of immersion. If the interaction tool is badly designed, such that the user's focus is more on the interaction tool than on the task itself, then this distraction may prevent immersion of the user in the application. In such a case, the reality distracts from the virtual reality experience. An intuitive and supportive interface can enhance the experience of the environment. Intuitive to people is everything they know by heart from real world experience. Breathing, walking, and, after a training period, driving a car are tasks which are performed without thought because these skills are stored in long term memory [Swe88, Swe94]. The cognitive load of these tasks is low, thus leaving space for people to simultaneously accomplish other tasks like thinking, learning, and solving problems, which require our consciousness, our working memory. Direct navigation in virtual environments with an interaction tool, for example, is a task which takes some time to learn, and thus may distract users from the actual application content. In contrast to this, head-tracked movement in space is highly intuitive.

Interaction Tasks

Interaction in virtual environments is, typically, direct interaction. The user initiates an action, controls motion or grabs objects, and the environment reacts immediately to this interaction. Mine [Min95] differentiates virtual environment interaction tasks as movement (motion through the world), selection, manipulation (translation of objects in space), and scaling (manipulating the size of objects). According to Bowman and Hodges [BH99], the main interaction tasks are

- navigation
- selection/picking
- manipulation[2]
- system control.

[2]This work will adopt the classification made by Bowman and Hodges, defining manipulation to be an arbitrary manipulation of an object.

Besides the general system control, navigation/motion can be seen as one of the most important interaction tasks. Before being able to properly pick or manipulate an object, the user needs to position himself appropriately to the object. For spatial exploration, the ability to move is a very important pre-requisite. Without being able to move around, the user cannot explore the space. In the next section we will discuss the different aspects of navigation we will later need in order to support exploration in virtual environments.

2.2 Navigation

Bowman et al. [BDHB98] define navigation in virtual environments as the complete process of moving through an environment. This process is divided into two parts: **wayfinding** (the cognitive decision-making process by which a movement is planned), and **travel** (the actual motion from the current location to the new location). Both parts are often interrelated. This definition is commonly used in the virtual environment research community (for example by Darken and Peterson [DP02]). Outside this community, wayfinding is defined to be the aggregate task of cognition and motion [Con01]. We will use the first definition, made by Bowman, though we think that both definitions are correct and useful in their respect, and that they both emphasize that navigation, especially for the purpose of exploration, is a complex task which has a strong cognitive element.

In [BH99], Bowman and Hodges define the three main navigation tasks as:

- **exploration**
  - without a specific target[3]
  - the target[3] is to build knowledge of environment
- **search**
  - naive search: position of target unknown
  - primed search: position of target known
- **maneuvering**
  - small movements to adjust viewpoint.

Exploration and search require large-scale travel through the environment. Travel addresses motion to a distant target, while maneuvering is a local process. Different interaction techniques are suitable for these tasks, which are discussed in the following sections.

Prior to travel for the purpose of exploration or naive search, some kind of "wayfinding" decision must be made as to where to move next. We will pick up on this as

[3]We prefer the use of goal instead of target. In our definition, a goal is the reason why a person explores an environment, while a target has a location in space.

we think that there potentially could be more to wayfinding for exploration than finding a way to a target.

2.2.1 Maneuvering

Maneuvering requires small local adjustments of the user's position. In immersive virtual environments, the most intuitive metaphors for local movement are walking and stretching or ducking under an object, actions which people would also do in real environments. These movements are enabled by head-tracking. By tracking the position and orientation of the head, the system knows the position of the user's eyes and can adjust the view accordingly. When the user moves, the eye-point of the view is also moved. As small movements of our head enhance our 3D perception in the real world, positional tracking creates an additional level of immersion in the virtual world. Depending on the virtual environment system, head-tracking can be used to view an object from both sides, or even walk around it, without actually modifying the position of the scene relative to the viewing system (the view platform, see Appendix D.2) itself. This interaction matches with peoples' real world behavior for this kind of task, thus is easy to do and does not distract the user. Therefore, head-tracking provides for the first and most intuitive interface in immersive virtual environments.

2.2.2 Travel

Once the user reaches the borders of the physical space of the virtual environment system, the scene itself relative to the physical space has to be moved. Travel is the process of moving from the current position to a target and, in general, is understood to be manually controlled motion. This means that the user controls the direction and the speed of motion. There are several travel metaphors and interfaces which help with this motion task.

Motion interfaces are categorized into active and passive interfaces [DM94]. Active interfaces, also called locomotion interfaces, require self-propulsion by the user. Examples of locomotion interfaces are "walking in place" or pedaling devices, foot platforms, and treadmills [DC97, Hol02]. Passive interfaces transport the user through the virtual environment without significant user exertion. A nice overview of passive travel techniques – here called viewpoint[4] manipulation – is given by Hand [Han97]. Some examples are the *Helicopter metaphor* (using a joystick) described by Brooks [Bro88], the *Scene in Hand metaphor*, the *Eyeball in Hand metaphor* (mapping of a 3D input device directly onto position and orientation of the viewpoint), and the *Flying Vehicle Control metaphor*, all described by Ware and Osborne [WO90]. Turner et al. [TBGT91] in their application and some recent

[4]The viewpoint describes the position and orientation of a user or camera in the virtual environment.

modelling packages let the user control the camera with a virtual camera object inserted in the scene. Stoakley et al. [SCP95] utilize the World-in-Miniature (WIM) technique. The last technique is an exocentric technique – the control is exercized from outside the virtual environment– while the others are done in an egocentric point of view inside the virtual environment.

2.2.3 Target-Based/Automated Travel

Most of the travel metaphors in literature are direct interaction methods in which the user directly controls the motion. This is interesting, as directly controlled motion requires considerable skill in controlling the interface and interaction tool. Already in 1995, Baker and Wickens [BW95] list the *put-me-there travel* as one of the main travel metaphors. It is also referred to as *automated travel.* In [Bow02], Bowman established the first guideline for designing travel techniques as:

”Make simple tasks simple by using target-based techniques.” Bowman [Bow02]

Surprisingly, there is very limited literature about the implementation of target-based travel techniques. Target-based travel requires the selection of a visible target by point-and-click or by selection from a list. Mackinlay et al. [MCR90] present a technique to move the user on a straight line with a speed logarithmically related to the distance to a target. VRML browsers provide for motion in a straight line to a clicked object or to a viewpoint selected from a list. Both systems do not enable collision prevention with objects on the way. This is not a major issue if visible targets were selected, as the straight line there provides for a collision-free path.

Until now, even this simple support for travel was not integrated into typical virtual environment systems and there is no application-independent approach to automated travel in the literature. Automated travel is one of the pre-requisites to guided exploration. Therefore, we will investigate into a method for automated travel, which is *travel from the current position to a single target.* We will define a list of requirements for a method which is capable of providing automated travel in current virtual environment applications.

Guidelines for Automated Travel

As stated before, interaction in virtual environments is normally direct interaction. User input is directly mapped to the behavior of the environment. Unexpected behavior like non-existent, too slow, or not anticipated response to interaction will disturb the user. Pausch et al. [PST$^+$96] found that users felt their illusion of presence in Disney’s Aladdin ride was destroyed when characters did not react to them. Unexpected response is experienced when things happen which are normally not experienced by people. For example, flying too close or even through objects is unexpected as well as visually disturbing and disorienting. This calls for some means of collision prevention in virtual environments. Another example of unexpected behavior is the concept of "beaming", which is not existent in our real-world experience. Therefore, instantaneous transitions to distant points in the scene (e.g. jumps, cuts, and teleportation) are not desirable in immersive virtual environments as they would largely disorient the user. This is supported by a user study by Bowman et al. [BKH97], who suggest that jumping disturbs the users, reduces the sense of presence, and diminishes spatial awareness. In [Bow02], Bowman emphasizes this again and asks for smooth transitions even if targets were selected on a map or with a World-in-Miniature, thus in an exocentric manner.

For travelling, this suggests that the motion should result in a continuous movement of the user (the view) through the virtual environment. If a distant target is specified by one of the above-mentioned methods, the application should be able to generate a path and move the user to this target. The same holds for non-immersive applications on a desktop PC, for example, if the goal is to keep awareness of the spatial context. Even in films, long established cinematographic rules have to be obeyed for utilizing cuts, in order to avoid confusing the viewer.

Therefore, if the system allows the abstract specification of travel targets through maps, lists, or queries (target-based interaction), it should also provide for a continuous, motion generation system and immediate response. Current virtual environment applications include dynamic applications, in which not only the viewpoint but also the scene itself changes frequently. Examples are simulations, applications which include autonomous avatars, and storytelling applications. Therefore, the system should be able to deal with dynamic, unpredictable environments. To ensure a general purpose system, the method should be independent of the application itself.

Requirements for Automated Travel

To conclude, the requirements for a system providing automated travel in virtual environments are:

1. immediate response to target definition
2. collision avoidance
3. continuous, smooth transition
4. dynamic scene capability
5. system independence

2.3 Exploration

To *explore*, according to the Merriam-Webster Collegiate Dictionary [Mer], has three meanings:

> **to explore**
> **1 a:** to investigate, study, or analyze : look into <explore the relationship between social class and learning ability> – sometimes used with indirect questions <to explore where ethical issues arise – R. T. Blackburn> **b:** to become familiar with by testing or experimenting <explore new cuisines>
> **2:** to travel over (new territory) for adventure or discovery
> **3:** to examine minutely especially for diagnostic purposes <explore the wound> intransitive senses : to make or conduct a systematic search <explore for oil>.

While the first definition describes exploring in general, the latter two are concerned with exploring a spatial environment. The Virtual Voyage application by Hong et al. is an example of definition **3**, as it assists in the exploration of computer tomography (CT) data of the human colon by an automated ride through the middle of the colon [HMK$^+$97]. Exploration in immersive virtual environments is best met by definition **2**, as typically the environment must be travelled in order to discover the content of the world.

2.3.1 Exploration in Virtual Environments

While travel itself is motion along a path to a well-defined target, exploration either does not specify a target at all, or it considers, driven by the goal for exploration, multiple, possibly interesting targets.

Exploration in virtual environments, as defined by Bowman and Hodges [BH99], is one of the three navigation tasks, and is either *a)* travel without a specific target, or *b)* the target is to build knowledge of the environment (recall Section 2.2). We feel that this definition needs to be modified in two ways. First, the meaning of the term *target* in this definition is unclear. Throughout most of Bowman's work, travel denotes the motion from a current position to a distant location. But in the definition above, the target location is unclear. We think that the term *goal* is more appropriate to use here. The goal is the reason a user explores an environment, or why targets are selected. It can be the wish to answer a question with the help of the environment, or to learn about a task. The goal is the pre-requisite of identifying one or multiple targets. Second, Bowman includes travel without a specific goal in his definition of exploration. In our opinion, there always exists a goal for exploration, even if it is as unspecific as "let me find out what is in here." Therefore, we define exploration as *the goal-based motion through the environment for the purpose of discovery or learning.*

To clarify this, we point out a major difference in travel and exploration:

Travel is a target-oriented motion. The focus is on arriving at the target as quickly as possible. The goal of motion is the target.

Exploration is goal-driven motion. In the context of the user's goal, there may exist one or multiple possibly interesting targets, which may be more or less interesting. When moving through this *interest landscape*, the goal field, the motion though the environment to the target is the goal itself. The focus is on browsing the space and viewing possible targets in their spatial context, or even identifying new targets by refining the goal.

Exploration is a highly interactive process, which we know from browsing the internet with search engines. An initial, broadly phrased query based on our goal may result in multiple links with high or low relevancy to the phrased question. Then, iteratively, the results are browsed and the query is refined, or the overall goal is changed (sometimes resulting from distractions). Each query results in an interest landscape of links (targets). For 3D spatial environments, the interest landscape is a field of spatially distributed targets with a degree of relevancy to the goal. We call this the **goal field** (see Section 1.1.2).

The question remains, how does the user identify the targets which match the goal. In virtual environments, this is normally done by directly controlling motion through the environment using one of the provided travel metaphors, and searching the space for targets. This may not lead to acceptable results, thus it makes sense to look into support for exploration.

2.3.2 Supported Exploration

Billinghurst and Savage [BS96] use a rule-based expert system to add intelligence to the user interface. They couple multimodal input (voice, gesture, body position), context, and the expert system to create an interface which is more powerful than the single components. It is used to find out what is meant by the user when he or she interacts in a specific way. They support interaction in general with the help of an expert system but do not contribute to navigation support.

Rickel and Johnson [RJ97] built a system for intelligent tutoring in virtual environments by providing an animated, pedagogical agent for virtual reality. This agent helps students learn to perform a physical, procedural task by demonstrating how to do it. In this approach, the focus is on teaching a "craft", but not on exploring a subject.

The first support in 3D navigation was proposed by Hanson and Wernert [HW97]. They state that there is a poor match between the goal of a navigation activity, the control device, and the skill of the user. They propose a unified mathematical framework for incorporating context-dependent constraints into the viewpoint generation problem. They rely on a system designer, who specifies his or her idea of directing a naive user's attention to those aspects of a scene which are needed to meet a chosen goal. They constrain the user-initiated control of an interaction device, in their example a mouse, to a useful motion in 3D space. The user travels the "guide manifold". At each position on this manifold, a "guide field" specifies all relevant scene-viewing parameters for this position. This was extended by Wernert and Hanson [WH99] to provide for a framework for assisted exploration with collaboration. They add avatars to the scene, which work as a guide or to represent other users. The guide avatar points out places of interest. The user's motion is constrained, as in their previous work, to simplify the motion task. This is the only approach which tries to incorporate assistance to exploration in virtual environments. It leaves some freedom to explore the space while providing assistance in where to look. As the constraint space is designer-specified, it is, in a sense, an interactive, self-exploitable presentation of the goal with its multiple targets. This approach does not allow a modification of the goal.

Supported Exploration Utilizing Existing Information Spaces

Our idea is to utilize existing information spaces (like the internet search engines or application knowledge bases) to provide support in the identification of targets and then move to these targets one-by one. Technically, this is defining the goal field and the moving in this goal field for the purpose of exploring it.

Providing for an interface to non-visual, complex information (symbolic information as opposed to geometric information [Str98]) adds the possibility of introducing higher-level knowledge into the environment, knowledge that is not visualized in the virtual environment application. For example, in a medical training system like the 3D puzzle by Ritter et al. [RPDS00] in which pieces of a foot have to be assembled in the correct way (see Section 7.1), a query like "show me all objects that need to be connected to the selected one" provides for information which otherwise would not exist in the environment itself.

We want to provide this information without modifying the environment for two reasons. First, the design guidelines for virtual environments listed by Stanney [Sta02] and partly discussed above indicated that for a true spatial experience, the environment should "behave" like reality does, thus not change in an artificial way. Second, we want to have a supportive system which can be utilized by arbitrary applications, as we want to extend existing applications with this possibility.

Requirements for supported exploration

We will now list the requirements for a supportive system for exploration which can provide guided exploration. When using a supportive system, the user should always feel that she is in control. At any time, the user should be able to

- re-query the information space

- take over complete navigation control by means of the standard interaction tool
- overlay the system motion with output of the interaction tool to change the resulting position or orientation
- stand back and let the system do the motion inside the goal field

Considering these demands, the requirements for guided exploration extend the requirements for an automated travel system considerably. Here, the automated motion is in a goal field and acts as a supportive system to the user, thus many interactive influences have to be considered:

1. immediate response to goal (re-)definition
2. collision avoidance
3. continuous, smooth transitions
4. dynamic scene capability
5. system independence
6. goal field: multiple targets with different degrees of relevancy
7. consideration of distance and relevancy
8. consideration of the history of visited targets
9. orientation in direction of targets
10. incorporation of interactive user input

2.4 Motion Generation Techniques

The previous sections listed requirements for automated travel and supported exploration, both of which rely on motion through a virtual environment. This section will inquire into known motion generation techniques, including a description of what is technically meant by motion in a virtual environment and a description of the different pre-calculated and interactively generated motion methods.

Motion of a camera is always motion along a path. This path may be pre-computed, or it may result retrospectively from the actual movement along the path. For the first case, path planning methods are utilized which generate the path and then execute the motion along the path. In the second case, the path only exists in the past, as it creates itself from the stepwise movement of the camera through the space. The motion is generated on-the-fly. This either requires rapid planning of one or multiple steps into the future or a method being reactive to current local and global influences.

Whether motion can be planned or has to be generated on-the-fly depends on the level of interactivity and the required response time of the application. Planning allows for the incorporation of cinematographic rules. In dynamic environments, without knowledge of the position of objects, planning is impossible, because the path only considers the positioning of obstacles at the time of path generation. Each movement of objects involves recalculation of the path. Reactive systems do not have much foresight, but nevertheless may show sufficient results for generating camera motion in real-time. The different approaches are discussed in this section. As the purpose of this work is to provide a system to support exploration which results in the presentation of objects in the context of a 3D world, applications for generating presentation animations are also mentioned. We first explain about motion in virtual environments and then introduce both pre-calculated and real-time motion generation techniques and their applications.

2.4.1 Motion in Virtual Environments

In a virtual environment, the user is represented by his or her eye-point. Moving the user through the virtual world, or moving the world such that specific objects come close to the user, is technically equivalent to moving the eye-point. The eye-point is part of the view, thus the camera (see Appendix D). Therefore, providing a support system for exploration requires a close look into view motion in virtual environments.

The virtual camera is an abstraction of a real camera (see Appendix C and D). It describes a pin-hole camera model. The projection function defines which parts of the 3D world are mapped to points in the projected 2D image. The projection function used in typical virtual environments is a perspective pinhole camera due to real-time rendering requirements. It requires a view position – the user's eye position – and a view frustum, represented by a truncated pyramid. The perspective frustum describes the 3D space that is rendered into the 2D image. It defines which objects are visible and in which order from front to back they are sorted. It emphasizes the importance of close objects by drawing them larger than distant objects.

In a virtual environment, motion can include motion of the view frustum, motion of the eye position of the user, or a combination of both. Rotating the view frustum moves objects inside the view. This concept also holds for 360° fully immersive projection systems if a main viewing direction is established. This is done by the design of the system or the user herself. Moving the view frustum through the 3D world brings objects closer to the user. By moving close to an object, the likelihood of unobstructed view on this object is largely increased.

A camera motion system which supports exploration should therefore consider the orientation of the camera, for moving objects inside the view frustum, and translation, for moving close to objects for unobstructed and prominent visibility.

2.4.2 Computer-Generated Motion

Two areas of research with a different focus are concerned with computer-generated motion. The first is concerned with finding a free path between two points in a environment with obstacles, thus a path from one point to a target. The focus of the second one is on rules for presentation of one or multiple targets, resulting in specific views onto the target. Both are presented in the following sections.

Travel between two points

Methods for travel between two points draw a lot from robotics. Latombe[Lat91] presents three general concepts for robot motion planning in which a start configuration and a goal are given, and a path to the goal is calculated while avoiding obstacles. The presented methods are called roadmap, cell decomposition, and potential fields.

Roadmap methods capture the connectivity of the free space in a network of one-dimensional curves called roadmap R. For path planning the initial and goal configurations are connected to R, and R is searched for a path between these points. The visibility graph [Nil69], Voronoi diagrams [ÓSY83], freeway nets [Bro83], and silhouettes [Can88] are examples of roadmaps.

Exact [Lat91, SSH87] and approximate [BLP83, LP81] cell decomposition methods decompose the free space into cells in such a way that a path between any two configurations can be easily generated. A connectivity graph is generated, and with this a continuous free path is computed.

Potential field methods, first introduced by Khatib [Kat86], discretize the free space into a fine rectangular grid. A particle moves through the grid under the influence of an attractive force introduced by the target and repulsive forces introduced by obstacles. It is these forces which generate a path. Compared with roadmap and cell decomposition, potential fields are more efficient and produce instantaneous motion, but may not always find a solution and may get stuck in local minima. Potential fields and their origin are discussed in detail in Chapter 3.

Considerable work has been carried out on methods for finding a free path between points x and y in a virtual 3D world. For example, Lavalle [Lav95] uses a game-theoretic framework for path planning. Bandi and Thalmann [BT98] divide the space into a 3D grid of uniform cells and, with the A* algorithm (a roadmap approach), compute the shortest path from x to y.

Presentation of targets

Another area of computer-generated motion research deals with the presentation of predefined targets under consideration of rules for good camera views and constraints stating which object has to be viewed in which manner.

Karp and Feiner [KF90, KF93] implemented ESPLANADE, a knowledge-based system for the automated generation of animated presentations. They have a separate action planner to pre-compute a script which is then used, together with a set of presentation goals, to generate the complete description of the animation for each frame.

Drucker and Zeltzer [DZ94] model the methods used by a film director: logic-based constraints are defined which govern good views on the scene. With this information, optimal camera positions for individual shots are calculated by solving small, constrained optimization problems. Then, with the A* algorithm, a path is generated which connects these positions.

All this research deals with previously specified targets and does not allow for interactivity. The calculation of the complete path is done in advance before the presentation can be started.

2.4.3 Interactive Computer-Generated Motion

He et al. [HCS96] extended the approach of Drucker and Zeltzer by providing additional idioms to camera modules. With this, a number of domain-dependent director's instructions are defined which are solved with a hierarchical, finite-state machine. He et al. select shots from a small set of possible camera specifications. They also have low-level camera modules which deal efficiently with geometric details. To meet the specifications of the constraints, small changes to the positions and actions of the virtual actors are introduced, thus changing the application. Actors may also be deliberately removed to clear the view. The application is required to behave in such a way that it is possible to predict the future actions of an "actor". In their example of a "party" application, a user can control actions of a guest through a high-level interface, providing actions like *talk, react, goto, drink,* and *lookat* by clicking an appropriate button. Though they generate the application in real-time, He at al. allow no direct interaction with the actors and no interactivity on camera level.

A different problem is addressed by Hong et al. in [HMK$^+$97]. In their Virtual Voyage, the exploration of computer tomography (CT) data of the human colon is assisted by an automated ride through the middle of the colon. The wall of the colon is extracted and skeletonized from the CT data. From this skeleton, the middle of the colon is calculated. The start and end of the automatic ride is predefined through the start and end of the colon. The movement is computed through a potential field, with the middle of the colon encoded as a path into the field. The user can manipulate the path with a mouse by clicking on a region of interest. A temporary target is then set at this point, and the camera is attracted to this region. In this application, the possible path through the colon is clearly defined after a preparation step, and the deviation introduced by the user is minimal. This application produces just-in-time camera data, but because of its very long initialization step and fixed overall start and goal, it is not suitable for supporting exploration under our requirements.

2.5 Summary

This chapter gave an introduction into virtual environments, their application, and systems. One of the major features of a virtual environment aside from the capability of providing for a spatial experience, is the "feeling of being part of the virtual environment". This is influenced by the system and the interaction facilities. The main interaction task is navigation because without motion, the space cannot be explored. Navigation is divided into a cognitive element (often called wayfinding) and travel. The three navigation tasks defined are exploration, search, and maneuvering. Exploration requires the identification of targets and the travel to these targets. Travel is the process of moving from a current location to a target. Several travel metaphors exist which mostly directly control the viewpoint. This is surprising, as one of the basic guidelines for designing travel techniques suggests using automated (target-based) travel techniques where possible. Automated travel is not integrated into typical virtual environments. We listed requirements in Section 2.2.3 which should be met by an system for automated travel in virtual environments. These include immediate response, collision avoidance, continuous and smooth transition, dynamic scene capability, and system independence.

Systems to support exploration are rare. We identified only one which assists the user in the motion, but does this by restricting the user motion through pre-defined constraints. We postulated a system to support exploration by integrating results of a query to an external information space. The resulting goal field, a list of targets and their relevancy to the goal introduced in Section 1.1.2, is then presented by moving the user to these targets. The requirements for a supportive system, defined in Section 2.3.2, extend the ones defined for the automated travel in Section 2.2.3. Additionally, full interactive influence of the results, motion in a goal field, the consideration of distance and relevancy, plus the history of presented targets are required.

Motion generation techniques were evaluated as to their capability to serve as motion generation systems for supporting exploration. The potential fields method is a real-time capable, local method for generating motion through an environment while avoiding obstacles in the scene. We need more than this, but will take this method as a starting point for our system. In addition to this method's real-time capability, our notion of a goal field fits nicely into the definition of a potential field.

In the following chapter, we will first develop a system for automated travel based on the potential field method. Collapsing the goal field to one target will result in automated travel. We then will extend this to develop a system which supports exploration.

3 Automated Travel using Potential Fields

Automatic, interactive camera motion for exploration support in virtual environments (i.e. guided exploration) requires a motion generation function which is capable of generating camera data in real-time. We start by developing a method and system which is capable of providing automated travel, a pre-requisite to support exploration. Automated travel is the automatic motion from a current position to a target location while avoiding collision with obstacles. The attributes of the motion generation function are the *geometry* of the scene's objects, the *camera data* (position and orientation) of the application, and the *target*. In interactive and dynamic applications, these attributes can change at any time, thus the the real-time requirement. Changes of the attributes may result from a transformation of the objects in the scene, a user initiated re-definition of the target, or a change of the camera data by the user or the application independent from the system's activity.

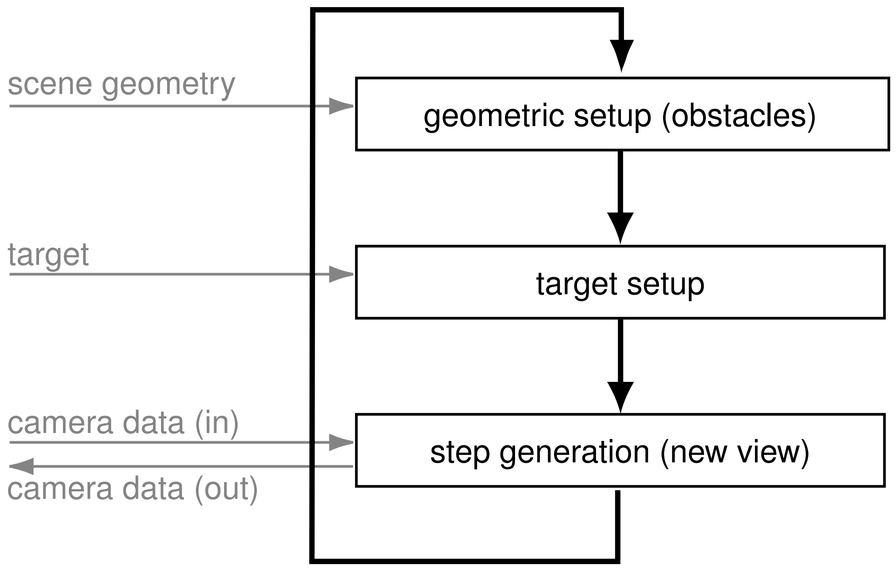


Figure 3.1: Overview of Process of the CubicalPath Method.

Our method for automated travel, which is implemented in the **CubicalPath system**[1], extends the real-time capable potential field method to a motion generation method which generates new camera data step by step. The process of generating one step involves three stages, illustrated in Figure 3.1 and further explained in this and the following chapters.

1. **Geometric setup of the potential field:** The system discretizes/voxelizes the geometric data into a 3D rectangular grid of cubes, the cube space. Objects are represented by the set of cubes which span the space of the object. This reduces complex geometric objects to a small subset of uniform cubes.

2. **Target setup of the potential field:** The target object receives a large attractivity value, while obstacles receive no attractivity. The system maps the attractivity of the objects to the attractivity of the corresponding cubes.

3. **Camera step:** The final encoded potential field stored in the cube space is evaluated at the position of the current camera. Attractive cubes generate an attractive force, while obstacles generate a repulsive force. According to these forces, the camera is moved to its new position, and the new camera data is distributed to the application.

After step 3, the process repeats.

The reminder of this section will introduce the basics of the potential field method and its utilization for automated travel. It will explain the required data structures and give the function for a basic view motion in a potential field with one target. It will introduce the three steps in the process diagram of Figure 3.1. This chapter assumes that a real-time capable voxelization method exists which is required for the geometric setup of the potential field.

Chapter 4 will extend the method introduced in this chapter to a fully interactive, motion generation system for supporting exploration which allows for multiple targets forming a goal field. It also allows for specific views. User input can change the camera position and the targets at any time.

In Chapter 5, two voxelization methods are introduced. A software-based voxelization method generates voxelized geometric data for every single object. A graphics hardware-based method provides for real-time voxelization of large dynamic environments.

3.1 Potential Fields

Potential field theory has its origin in theoretical physics [AF00]. It describes the behavior of particles in electrostatic fields and the Newtonian attraction between

[1]The name CubicalPath is derived from the main data structure used, a three dimensional set of cubes, which represents the discretized version of the environment, and the fact that a path is generated. Different to other methods, this path only becomes visible retrospectively.

masses. The scalar potential fields and their derivatives which are vector fields have some mathematical and practical features that will be illustrated utilizing the physics and mathematics of electrostatics. Then, their usage for target finding and obstacle avoidance in robotics is briefly described. The range of applications for potential fields is then extended to real-time camera motion.

3.1.1 Electrostatics

The *electric potential* V (measured in Volt) is a scalar field indicating the "height" – the potential – for every point in the field (see Figure 3.2). If a charged particle q is inserted at point $\overrightarrow{p}$ into this potential field, it has the potential energy

$$E(\overrightarrow{p}) = qV(\overrightarrow{p}). \tag{3.1}$$

If the particle is moved from point $\overrightarrow{p_1}$ to point $\overrightarrow{p_2}$ with $\Delta\overrightarrow{p} = \overrightarrow{p_2} - \overrightarrow{p_1}$, then the work

$$W(\Delta\overrightarrow{p}) = E(\overrightarrow{p_2}) - E(\overrightarrow{p_1}) = q(V(\overrightarrow{p_2}) - V(\overrightarrow{p_1})) \tag{3.2}$$

is done. The particle gains or looses potential energy, thus move up or down in the potential field. If the potential at the target position is equal to that of the start position, then the work done is zero, even though the particle may have moved up and down the field along the way.

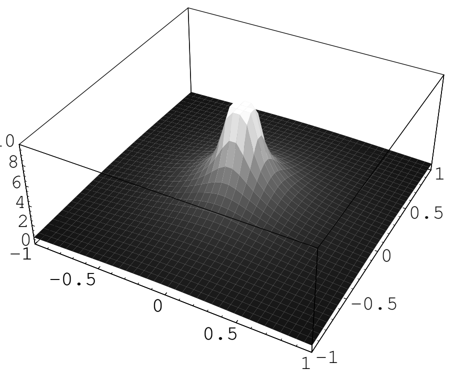


Figure 3.2: Electrostatic Potential Field of a Point Source. This figure shows the potential values depending on their distance to the center of the charge.

The *electric field* is the first derivative of the potential field:

$$\overrightarrow{\epsilon}(\overrightarrow{p}) = \text{grad } V(\overrightarrow{p}) = \overrightarrow{\nabla}_{\overrightarrow{p}} V(\overrightarrow{p}). \tag{3.3}$$

$\overrightarrow{\nabla}_{\overrightarrow{p}}$ is the gradient with respect to $\overrightarrow{p}$. The electric field is a vector field, indicating the gradient of the potential field for each point in the field. A charged particle q inserted into this field will experience a directed force that depends on its own charge and the electric field at the position

$$\overrightarrow{F}(\overrightarrow{p}) = q\overrightarrow{\epsilon}(\overrightarrow{p}). \tag{3.4}$$

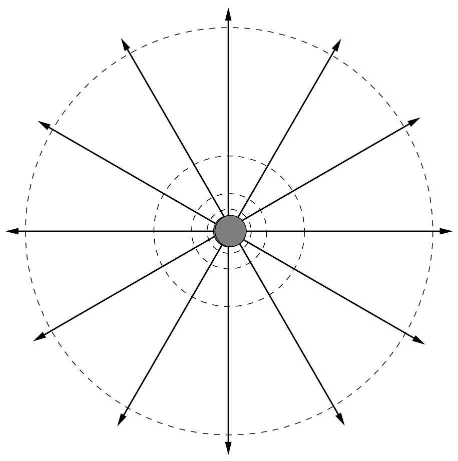


Figure 3.3: Electrostatic Field of a Point Source. This figure shows equipotential lines (dashed) and lines of force (solid).

An electrical field is generated by a potential difference between two positions in space. For example, a point charge at position $\overrightarrow{q}$ (as shown in Figure 3.3), generates a spherical potential field with electrical field vectors pointing away from the charge. This potential field is illustrated in 3D in Figure 3.2. The point source builds a potential wall around itself which decreases with the distance. Near the charge, the equipotential lines are closer together, as the potential field decreases with $\frac{1}{|\overrightarrow{r}|} = \frac{1}{|\overrightarrow{p}-\overrightarrow{q}|}$:

$$V_{point\_charge}(\overrightarrow{p}) = \frac{q}{4\pi\epsilon_0|\overrightarrow{p}-\overrightarrow{q}|} = k\frac{q}{|\overrightarrow{p}-\overrightarrow{q}|} = k\frac{q}{|\overrightarrow{r}|}. \tag{3.5}$$

The term $\frac{1}{4\pi\epsilon_0}$ is constant and can be replaced by the constant k. The electrical field of a point charge, the derivative of V_{point_charge}, is

$$\begin{aligned}
\overrightarrow{\epsilon}_{point\_charge}(\overrightarrow{r}) &= \overrightarrow{\nabla}V(\overrightarrow{r}) = \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix} V(\overrightarrow{r}) = \begin{pmatrix} \frac{\partial V(\overrightarrow{r})}{\partial x} \\ \frac{\partial V(\overrightarrow{r})}{\partial y} \\ \frac{\partial V(\overrightarrow{r})}{\partial z} \end{pmatrix} && (3.6)\\
&= -k\frac{q}{|\overrightarrow{r}|^3}\begin{pmatrix} x \\ y \\ z \end{pmatrix} \text{ (see Appendix A for derivation)} && (3.7)\\
&= -k\frac{q\overrightarrow{r}}{|\overrightarrow{r}|^3}\\
&= -k\frac{q}{|\overrightarrow{r}|^2}\frac{\overrightarrow{r}}{|\overrightarrow{r}|}. && (3.8)
\end{aligned}$$

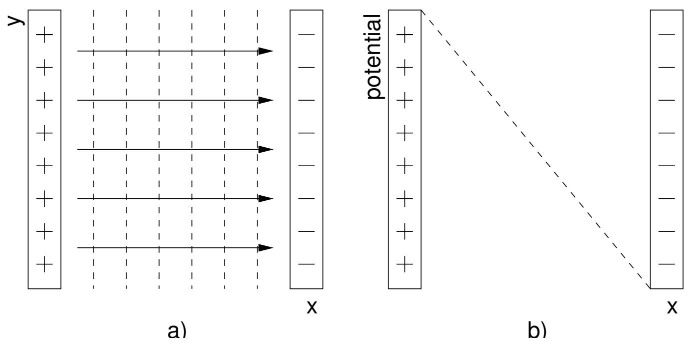


Figure 3.4: Electrostatic Field between Two Different Charged Plates. Figure a) shows equipotential lines (dashed) and lines of force (solid). Figure b) shows the decreasing potential energy (dashed) between the positive and the negative charged plate.

A linear field, as in Figure 3.4, can be generated by charging two metal plates, one negative and one positive. A field appears between the two plates, with forces perpendicular to the plates pointing from the positive charged plate to the negative. The equipotential lines are orthogonal to the force lines, thus parallel to the plates. Any positive charged particle inserted into this field will move in the direction of the negative plate. The larger the potential difference, the larger the force acting on the charge.

A particle q at position $\overrightarrow{p}$ in an electrical field will experience the force $\overrightarrow{F}(\overrightarrow{p})$ and the acceleration $\overrightarrow{a}(\overrightarrow{p})$:

$$\overrightarrow{F}(\overrightarrow{p}) = q\overrightarrow{\epsilon}(\overrightarrow{p}) \tag{3.9}$$

$$\overrightarrow{a}(\overrightarrow{p}) = \frac{q}{m}\overrightarrow{\epsilon}(\overrightarrow{p}). \tag{3.10}$$

Two particles, one charged negative and one charged positive, (see Figure 3.5 b) will produce a force between them, in which the lines of force point from the positive charged particle to the negative charged one. In general, if two charges interact with each other, the force between them depends on the distance vector $\overrightarrow{r}$ between the two and is

$$\begin{aligned}\overrightarrow{F}(\overrightarrow{r}) &= q\overrightarrow{\epsilon}(\overrightarrow{r}) \\ &= -qk\frac{q'}{|\overrightarrow{r}^2|}\frac{\overrightarrow{r}}{|\overrightarrow{r}|}.\end{aligned} \tag{3.11}$$

This force depends on the charges and their inverse quadratic distance. An example is illustrated in Figure 3.5 a) and b).

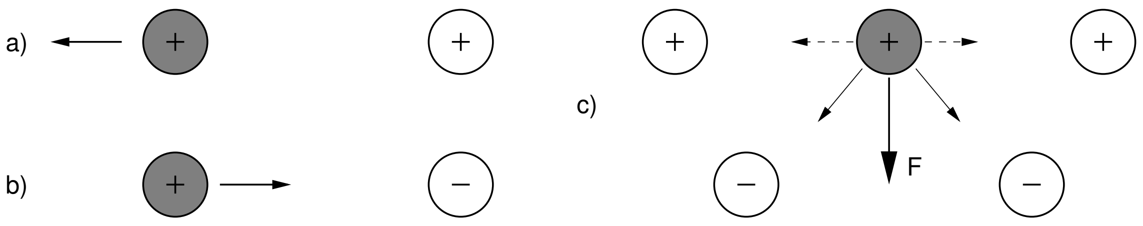


Figure 3.5: Force Generated by Multiple Point Charges. This figure shows a free-floating, positive charged particle (grey) under the influence of other charged particles. In case a) the particle is repulsed by the equally charged, fixed particle. In case b) the particle is attracted by the differently charged particle. Case c) shows the compound influence of multiple particles, two positive and two negative charged ones. The resulting force vector points between the two negative charged, attracting, particles. In this example, the forces of the repulsing positive charged particles cancel each other out.

Multiple charges

For multiple point charges, all potential fields are added to form the final potential field (see Figure 3.6):

$$V(\overrightarrow{p}) = k\frac{q_1}{|\overrightarrow{r}_1|} + ... + k\frac{q_n}{|\overrightarrow{r}_n|} = k\sum_{i=1}^{n}\frac{q_i}{|\overrightarrow{r}_i|} \tag{3.12}$$
$$\text{with } \overrightarrow{r}_i = \overrightarrow{p} - \overrightarrow{p_i} \text{ ;} \overrightarrow{p_i} = \text{position of charge } q_i$$

By using Equations 3.8 and 3.11, the electrical field at position $\overrightarrow{p}$and the force on a particle q at position $\overrightarrow{p}$for multiple point charges are:

$$\overrightarrow{\epsilon}(\overrightarrow{p}) = \overrightarrow{\nabla}V(\overrightarrow{p}) = \overrightarrow{\nabla}k\sum_{i=1}^{n}\frac{q_i}{|\overrightarrow{r}_i|} = k\sum_{i=1}^{n}\overrightarrow{\nabla}\frac{q_i}{|\overrightarrow{r}_i|} = -k\sum_{i=1}^{n}\frac{q_i}{|\overrightarrow{r}_i|^2}\frac{\overrightarrow{r}_i}{|\overrightarrow{r}_i|}. \tag{3.13}$$

$$\overrightarrow{F}(\overrightarrow{p}) = q\overrightarrow{\epsilon}(\overrightarrow{p}) = -qk\sum_{i=1}^{n}\frac{q_i}{|\overrightarrow{r}_i|^2}\frac{\overrightarrow{r}_i}{|\overrightarrow{r}_i|} \tag{3.14}$$

If a free-floating, positive charged particle is inserted into the compound potential field, it will move in the direction of the steepest negative gradient at that point, thus "downhill" into the next minimum of the potential field. This is because the force onto this particle points in the direction of the lower potential, as a result of Equation 3.11. Figure 3.5 c) shows the resulting forces calculated for a specific point in the force field.

This behavior of potential fields can be used to describe a landscape of attractive and forbidden positions in space. The first applications that made use of this principle were in the field of robotics.

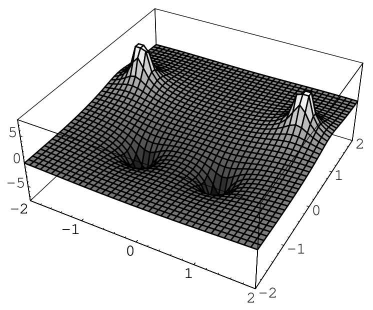


Figure 3.6: Potential Field Generated by Multiple Point Charges. This figure shows the potential field generated by two positive (forming peaks) and two negative charged particles (forming wells). A particle inserted into this field will "roll downhill" into the next well.

3.1.2 Robotics

One of the tasks in robotics is to find a free path to a target through an environment. As discussed in Section 2.4.2, there are three fundamental methods which can be used. The first two methods, roadmap and cell decomposition, generate a complete path to a target before the robot starts to move. Complete knowledge of the environment is needed, and a change in the environment requires recalculation of the path. The potential fields method is a much faster, real-time capable, local method which step-by-step generates the next position of a robot for the current environmental situation. The path evolves, as the robot moves to its target. By evaluating the current field, an instant answer to where to move is generated. This method was first introduced by Khatib [Kat86]. A thorough discussion to the usage of potential fields in robotics can be found in Latombe [Lat91]. This section gives a brief introduction.

In the potential field method, a robot $\mathcal{A}$ moves in its configuration space $\mathcal{C}$ as a point under the influence of an artificial potential field U. The configuration space is introduced to map the motion planning for a multi-dimensioned object into the problem of motion planning for a point. It is also used to map obstacles into this space. Each configuration q in $\mathcal{C}$ states, if the robot is allowed to move into this configuration considering all restrictions to its spatial and behavioral attributes.

The field function U can be defined over the free space as the sum of an attractive potential pulling the particle to the target, and a repulsive potential pushing the particle away from obstacles. At each iteration, the artificial force at position $\overrightarrow{p}$

$$\overrightarrow{F}(\overrightarrow{p}) = -\overrightarrow{\nabla}U(\overrightarrow{p}) \tag{3.15}$$

is the direction of movement.

The artificial functions for attractive and repulsive configurations are chosen, such that they generate the best motion behavior. In the presence of one target, the attractive potential field $U_{att}(\overrightarrow{p})$ can, for example, be defined as a parabolic well

$$U_{att}(\overrightarrow{p}) = \xi\rho_{target}(\overrightarrow{p})^2 \tag{3.16}$$

where ξ is a scaling factor and $\rho_{target}(\overrightarrow{p})$ denotes the Euclidean distance to the target. This function has a minimum at the target position. The robot will be "attracted" by this minimum and move towards it until it comes to a rest at the deepest point of the well, the position of the target.

The repulsive force of an obstacle should effect the particle only if this particle is getting close to some threshold. The repulsive potential function can be defined as

$$U_{rep}(\overrightarrow{p}) = \begin{cases} \eta(\frac{1}{\rho_{obst}(\overrightarrow{p})} - \frac{1}{\rho_0})^2 & \rho_{obst}(\overrightarrow{p}) \leqslant \rho_0 \\ 0 & \rho_{obst}(\overrightarrow{p}) > \rho_0 \end{cases} \tag{3.17}$$

with η being a positive scaling factor, $\rho_{obst}(\overrightarrow{p})$ the distance to the obstacle, and ρ_0 the distance of influence.

In robotics application, there usually exists one global target but multiple obstacles. In the presence of multiple obstacles and targets, each of their artificial fields are summed. Overlaying attractive and repulsive potential functions as described above can result in local minima at positions other than targets. This is the major drawback of the potential field approach for motion generation. A situation, in which the robot or the camera gets stuck in a local minimum must be detected and dealt with. Methods for escaping local minima use Brownian movement for catapulting the robot out of the minimum or temporarily employing one of the path planning methods to search for a suitable path.

To generate the configuration space for a simple point robot, the geometric environment is discretized into a fine, regular grid. Any configuration q in the configuration space holds the information, if an obstacle and/or a target is inside this sub-space.

$\mathcal{A}$ may be a robot or, as in real-time camera motion, a camera.

3.1.3 CG Applications

Potential fields can be used to encode a path, a tunnel into a 3D data. This was done by Hong et al. in [HMK+97]. In their Virtual Voyage, the exploration of CT data of the human colon is assisted by an automated ride through the middle of the colon. The movement is computed through a potential field algorithm with the middle of the colon encoded as a path into the field. This application was explained in Section 2.4.3.

For his intelligent camera control, Drucker [Dru94] implemented a path planning method based on the A* search algorithm and pre-computed global navigation functions. In case of dynamic environments in a closed space, their method generates a repulsive gradient field using a distance map of the space. They overlay this with an attractive field in which the encoded target destination is in the minimum. Then, a breadth-first search explores the entire grid space. The results are then used to generate the final path by fitting a spline through the resulting points. Though this method is faster than running their complete algorithm, it still has to generate a complete path inside the space and involves several stages of path calculation. Also, this method does not take the direction of view into account, an important issue which we will explain later.

3.2 Camera Motion Utilizing Potential Fields

The approach to motion generation used by the CubicalPath system generates camera data – position and orientation – in real-time by combining an abstract description of the environment and the local, working, real-time capable potential field method. The first two stages of the process (see Figure 3.7) initialize the potential field which is then used in the third step for calculating new camera data.

This section introduces all three steps of the CubicalPath process required to build a working server system for camera motion. The control parameters in this figure – like objects, targets and the current camera – are set by the client application.

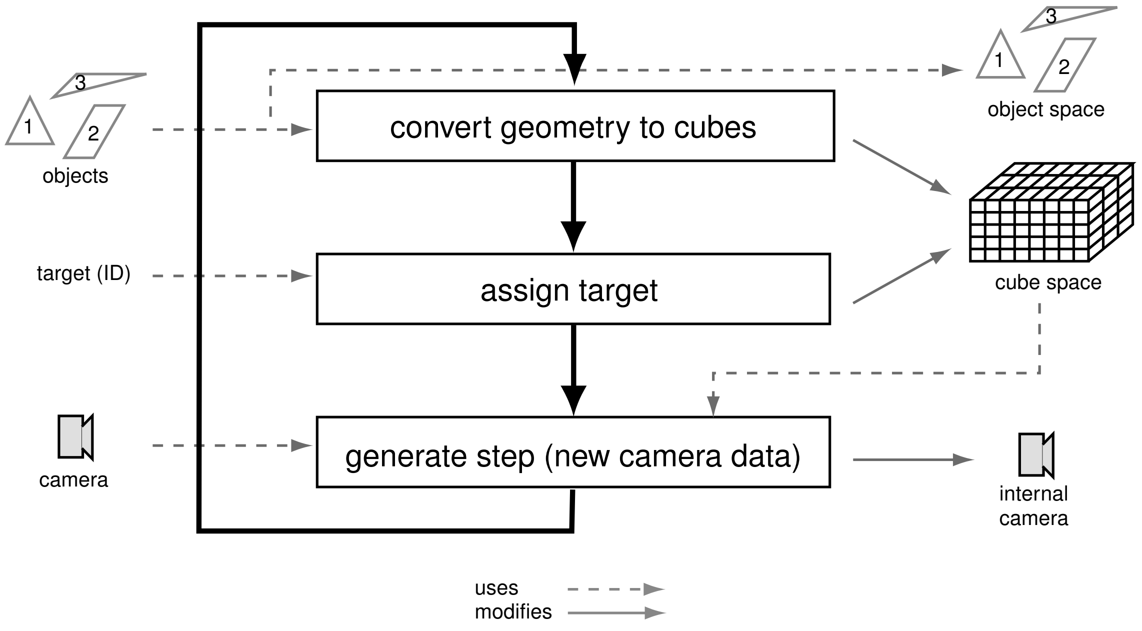


Figure 3.7: Process Diagram Including Data Structures.

3.2.1 Geometric Setup

Before the CubicalPath system can set up the potential field, the geometric objects need to be collected from the client application and stored independently. This space, which consists of the geometric objects, forms the **object space** (see Figures 3.7 and 3.8). Technically, this means that each object in the scene is administered as a separate entity with a unique ID. The object space represents the current visualized scene of the client application.

For building the potential field, the geometric scene is transformed into a regular three-dimensional grid – an array of cubes. This array of uniform cubes forms the **cube space**[2]. The resolution of the cube space is defined by the client application and depends on the required granularity of the scene. A coarse resolution requires less computation, while a fine resolution may resolve small passages between objects. Examples of the setup parameters are given in Section 7. The cube space defines the space in which obstacles and targets can be specified. Its extension is normally set to the bounding box of the scene.

The CubicalPath system stores information to establish a bilateral connection between the *cube space* and the *object space*. The attributes, stored with each object, control the behavior of the cube space by mapping control input of the client application (e.g. the object's attractivity) to the according cubes. For this, each object stores the identifying ID from the client application and the list of all cubes occupied by this object. Each cube can access all information about the original geometric scene inside the space it occupies via its list of objects. The parameters of the objects, cubes, and their bilateral connection are illustrated in Figure 3.8.

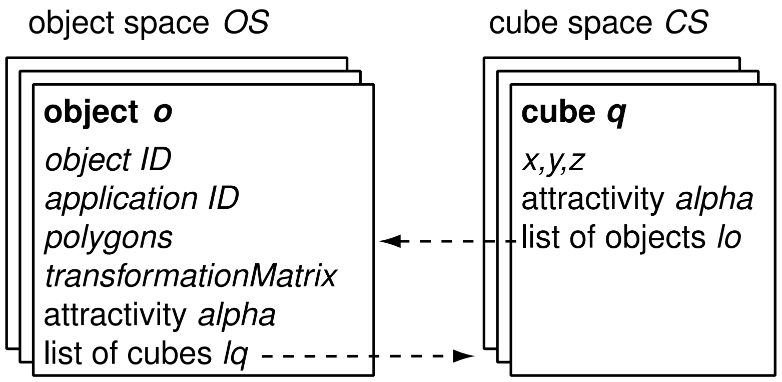


Figure 3.8: Object Space and Cube Space.

The geometric setup of the cube space requires the discretization/voxelization of the objects' geometry to retrieve the occupied cubes. Section 5 introduces methods to generate voxelized – discretized – representations of the geometric space while also retrieving object information for each of the resulting voxels. Both methods are able to identify multiple objects in one voxel. This is required when multiple small objects occupy the same cube, as shown by objects 2 and 3 in Figure 3.9.

With this setup of the cube space, the original contents of each cube at each position in space are known. Complex geometry can be discarded while the necessary information is preserved in the cubes.

[2]Technically, the cube space is the result of voxelization and is composed of voxels. We chose the word *cubes* and *cube space* to make clear that each cube has an extension, holds additional data structures, and that its prime purpose is not visualization.

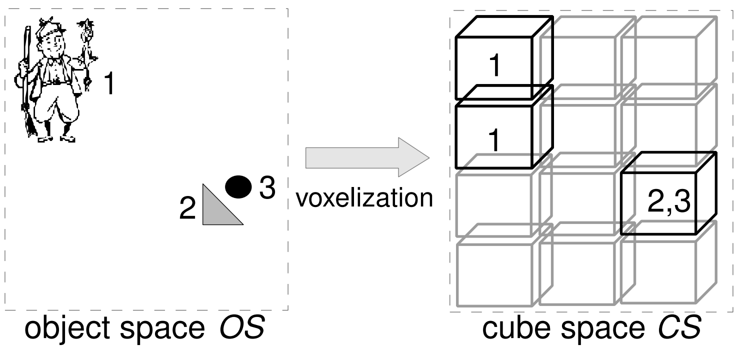


Figure 3.9: Voxelization of Objects in Object Space Forms Cube Space.

Repulsive Field Functions

Each element (cube) in the cube space which contains geometry is considered to be an obstacle to the camera. To avoid the camera getting close to obstacles or penetrating them, each cube which is an obstacle generates a repulsive field if the camera gets nearby. The CubicalPath system uses the following function, based on [Kat86], where $\overrightarrow{q}$ denotes the position of the obstacle and $\overrightarrow{d}=\overrightarrow{p}$-$\overrightarrow{q}$:

$$U_{rep}(\overrightarrow{p}) = \begin{cases} \eta_{rep}\left(\frac{1}{|\overrightarrow{d}|} - \frac{1}{d_0}\right)^2 & |\overrightarrow{d}| \leqslant d_0 \\ 0 & |\overrightarrow{d}| > d_0 \end{cases} \tag{3.18}$$

This function is zero outside the distance of influence d_0, as the term $\frac{1}{d_0}$ adjusts the function in such a way that it is zero at a distance equal or lager than d_0. The term $\frac{1}{|\overrightarrow{d}|}$ creates a function which grows towards the cube for a distance smaller than d_0. This definition of a field for obstacles results in a local field. It is designed such that it influences the calculation only in the case that the camera is close to the obstacle. This requires fewer calculations and results in less noise in the compound field.

The resulting fields of every cube forming an obstacle could now be assigned to each position of the cube space to encode the complete potential field as in Equation 3.12, and as used by Hong et al. [HMK+97] and Drucker [Dru94]. Then, the field is evaluated at the desired camera position, and the direction of the steepest negative gradient, its first derivative, is calculated. For this, any change of geometry would require a recalculation of the potential field for every position in the cube space.

Cost of using potential functions

Setup:	proportional to size of cube space (x*y*z) and the number of cubes forming an obstacle (no.of.obstacleCubes)
Step calculation:	constant (time required for calculating the derivative of the field at current camera position)

An equivalent result is achieved by first calculating the derivatives of the potential functions and then adding the resulting force vectors for the camera position similar to Equation 3.14. With the derivative of the potential field function for $|\overrightarrow{d}| \leqslant d_0$ in Equation 3.19, the vector field in Equation 3.20 is calculated (for the derivative see Equation 3.8).

$$\nabla_{\overrightarrow{d}} \eta_{rep} \left(\frac{1}{|\overrightarrow{d}|} - \frac{1}{d_0} \right)^2 = 2\eta_{rep}(\frac{1}{|\overrightarrow{d}|} - \frac{1}{d_0})(-1)\frac{1}{|\overrightarrow{d}|^2}\frac{\overrightarrow{d}}{|\overrightarrow{d}|}. \tag{3.19}$$

$$\overrightarrow{\epsilon}_{rep}(\overrightarrow{p}) = \begin{cases} \gamma_{rep}(\frac{1}{|\overrightarrow{d}|} - \frac{1}{d_0})\frac{1}{|\overrightarrow{d}|^2}\frac{\overrightarrow{d}}{|\overrightarrow{d}|} & |\overrightarrow{d}| \leqslant d_0 \\ 0 & |\overrightarrow{d}| > d_0 \end{cases} \tag{3.20}$$

$$\text{with } \gamma_{rep} = -2\eta_{rep}$$

The resulting force function on a particle with charge q is presented in Equation 3.21. This is used in Equation 3.22 for calculating the force of multiple charges q_i at position $\overrightarrow{q_i}$ on a particle q at position $\overrightarrow{p}$.

$$\begin{aligned} \overrightarrow{F}_{rep}(\overrightarrow{p}) &= q\overrightarrow{\epsilon}_{rep}(\overrightarrow{p}) \\ &= q\begin{cases} \gamma_{rep}(\frac{1}{|\overrightarrow{d}|} - \frac{1}{d_0})\frac{1}{|\overrightarrow{d}|^2}\frac{\overrightarrow{d}}{|\overrightarrow{d}|} & |\overrightarrow{d}| \leqslant d_0 \\ \overrightarrow{0} & |\overrightarrow{d}| > d_0 \end{cases} \end{aligned} \tag{3.21}$$

$$\overrightarrow{F}_{multipleRepCharges}(\overrightarrow{p}) = q\sum_{i=1}^{n} \overrightarrow{\epsilon}(\overrightarrow{q}_i) = \sum_{i=1}^{n} \overrightarrow{F}_{rep}(\overrightarrow{q}_i) \tag{3.22}$$

These derivatives can be made known to the system as the artificial potential functions are fixed and thus, their according force functions are fixed. This approach is used by the CubicalPath system. At the stage of the geometric setup, the field is analytically defined by the repulsive functions and the position of the fields' sources. These are parameter to the repulsive force functions. This approach largely reduces the setup time which is crucial in dynamic environments as a new setup may be required in each step:

Cost of using force functions

Setup:	proportional to no.of.obstacleCubes
Step calculation:	proportional to no.of.obstacleCubes

3.2.2 Target Setup

The second step is the setup of the target in the field. The target is supplied by the client application, for example, through a user selection.

Attractive Field Functions

The behavior of the camera is controlled by the attraction function, which is generated by the target. It is desirable that this function effects the whole field, and has a stronger influence close to the target to differentiate between near and distant targets. This becomes interesting in the context of multiple targets which are later used to support exploration. We define the potential function for an attractive cube at position $\overrightarrow{q}$ as

$$U_{attr}(\overrightarrow{p}) \quad = \quad \gamma_{attr}\alpha(\ln(|\overrightarrow{d}|) + |\overrightarrow{d}|) \tag{3.23}$$

where α is the attractivity of the cube and $|\overrightarrow{d}| = |\overrightarrow{p} - \overrightarrow{q}|$ is the distance to the cube. The attractivity α is set to 1 for the purpose of automated travel. The attractivity value will later be used to differentiate between more or less interesting objects in the context of exploration. The factor γ_{attr} controls the overall influence of attractive potentials to the compound potential function.

The according force on a particle q is:

$$\begin{aligned}
\overrightarrow{F}_{attr}(\overrightarrow{p}) &= q\nabla_{\overrightarrow{d}} U_{attr}(\overrightarrow{p}) \\
&= q\nabla_{\overrightarrow{d}} \gamma_{attr}\alpha(\ln(|\overrightarrow{d}|) + |\overrightarrow{d}|) \qquad (3.24) \\
&= q\gamma_{attr}\left(\alpha\frac{1}{|\overrightarrow{d}|}\frac{\overrightarrow{d}}{|\overrightarrow{d}|} + \alpha\frac{\overrightarrow{d}}{|\overrightarrow{d}|}\right) \\
&= q\gamma_{attr}\left(\frac{\alpha}{|\overrightarrow{d}|} + \alpha\right)\frac{\overrightarrow{d}}{|\overrightarrow{d}|}. \qquad (3.25)
\end{aligned}$$

The function $\frac{\alpha}{|\overrightarrow{d}|}$ affects the entire field. The smaller the value of $|\overrightarrow{d}|$, the larger the results of this function. The value α adjusts this function to generate smaller values for smaller α. For $\alpha = 0$, the result of the function becomes 0. The term $\frac{\alpha}{|\overrightarrow{d}|}$ converges to 0 at infinity. With the addition of α, this function is raised to converge at α, resulting in an influence proportional to the attractivity of the cube. The overall influence of the attractive forces to the compound force vector is controlled by γ_{attr}. For multiple charges q, thus multiple cubes, the resulting force at position $\overrightarrow{p}$ is

$$\overrightarrow{F}_{multipleAttrCharges}(\overrightarrow{p}) \quad = \quad \sum_{i=1}^{n} \overrightarrow{F}_{attr}(\overrightarrow{q}_i) \tag{3.26}$$

3.2.3 Step Generation - Camera Position

In each iteration, the new position of the camera is based on the compound attractive and repulsive fields. The field – the sum of all attractive and repulsive fields – is evaluated at the current position of the camera. In the potential field, the camera moves from the current position in the direction of the steepest negative gradient. Visually, this means that the camera rolls down-hill into the next minimum.

Instead of working directly with the potential function, the CubicalPath system uses the force functions, as defined in Section 3.2.1, which lead to the same results. We set q – the charge of the particle, in this the camera moving in the field – to 1.

Algorithm

The camera is a particle under the influence of the compound field of attractive and repulsive functions. The CubicalPath system uses the force functions to calculate the direction of movement as in Algorithm 3.1, using the Equations 3.22 and 3.26. These functions use the position and the attractivity information of each cube q in the cube space.

Algorithm 3.1: Calculation of the Direction of Motion of the Camera

```
for all repulsive cubes q
   repulsive_force_vec += calculate_vector(repulsive_force_func, camera_pos, q);
   noOfRepulsiveCubes ++;

repulsive_force_vec = repulsive_force_vec*rep_factor/noOfRepulsiveCubes

for all attractive cubes q
   attractive_force_vec += calculate_vector(attractive_force_func, camera_pos, q);
   noOfAttarctiveCubes ++;

attractive_force_vec = attractive_force_vec*attr_factor/noOfAttractiveCubes;

directionOfCamera = repulsive_force_vec + attractive_force_vec;
```

Motion Speed

The length of the resulting force vector and the time required for one iteration determine the velocity, $v = a * t$ (a is the length of the force vector), of the camera motion. Normally, the force causes a change in the current velocity and direction of motion. This requires the implementation of a physical motion model which would need the definition of gravitational and friction forces. We use this resulting velocity by multiplying it with the time required for the calculation and adding it to the old position.

The velocity of the camera is a crucial parameter for the perception of the motion. If the camera jumps, moves too fast or too slow, the user may feel uncomfortable. Thus, the speed of the camera should be controlled such that the camera moves at a pleasant speed which is adjusted to the requirements of the user and the application. For this, the maximum speed is controlled by the parameter v_{max} which cuts the length of the computed $directionOfCamera$ vector in Algorithm 3.1 if it gets larger than v_{max}/t. If the camera moves into a local minimum, it will automatically decrease its speed until it stops, as the resulting force on the camera gets smaller. By this, the potential field approach smoothly adjusts the speed if it is less the v_{max}. The overall speed in this range $[0..v_{max}]$ can be adjusted by manipulating the attractivity factor γ_{attr} in Equation 3.24.

Minima/Reliability/Robustness

If the speed of the camera becomes zero there may be several reasons. First, no attractive cube is in the cube space CS, thus there is no target. Second, the camera moved into the minimum formed by a target, and has reached its destination. Third, the camera moved into an unwanted local minimum of the compound potential field. This situation is detected by evaluating the distance to the target location. If there is a target defined and it is not yet reached, then the motion generation function has run into an unwanted local minimum. This is a rare situation in sparse environments with convex objects, but is more likely to occur in cluttered or complex environments like architectural models and mazes [Kat86]. To move out of an undesired local minimum, local methods like Brownian motion, random walks [Pap65] or randomized path planning [BL89] can be utilized. These will fail for complex tasks, however, because of their local perspective. In complex environments, a global path planning is necessary. Because the environment is already discretized, a straight-forward solution would be the use of the A* algorithm [Lat91] to search the environment for the shortest path to the target.

We chose the potential field method for its real-time performance in dynamic scenes and its ability to handle interactive input of target and camera data. These requirements do not allow for expensive path planning and search methods. Our method does not address complex environments. In the rare case of an unwanted stop of the camera in cluttered environments with convex objects, we rely on the user, who at all times holds a navigation tool in his or her hand, to move out of this minimum.

3.2.4 Step Generation - Camera Orientation

For real-time camera motion, especially if employed to support the presentation of an environment, the *visual task* is the major goal for the supporting system. It is vital for presentation to show meaningful content in the field of view. The motion itself is only a means of transporting the user to the requested objects or information. For this, we orient the camera such that the target is in the middle of the view. This is typically done with a "lookat" function [WNDO99]. This function generates the view depending on the camera position and the target to be looked at. The resulting direction of view will not be the direction of movement, if the direction of motion is not on a straight line to the target.

Rotation Speed

Just as we enforce a speed limit for the motion, it is necessary to enforce a maximum angular velocity for the rotation. Changes in the view have a much larger impact on the perception of motion than changes in the position of the camera. To avoid confusing the user, to keep spatial awareness and to reduce motion sickness, abrupt changes of direction have to be avoided. When a new target is to be moved in the view, the camera should slowly turn to the new target. Therefore, the angular velocity is restricted. For the virtual environment applications described in Section 7, we found that the angular velocity should not exceed 20° per s. To enforce the maximum angular velocity $\omega_{angular}$, the camera rotation is adjusted such that it rotates not more than θ_{max} in the direction of the calculated rotation. θ_{max} depends on the time for the current iteration and maximal angular velocity: $\theta_{max} = \omega_{angular} * t_{iteration}$. The algorithm uses the direction vector for the camera viewing direction. The calculation uses quaternions [Sho85] and is illustrated in Algorithm 3.2.

Algorithm 3.2: Enforcement of Angular Velocity

```
//angle between the two in radians
dotValue = oldDir.dot(newDir);
angle = fabs(cos(oldDir.dot(newDir)));

if (angle > enforcedAngularVelocity*timePerInteration ){
  float tslerp = enforcedAngularVelocity/angle *timePerInteration;
  oldMatrix.makeVecRotVec(fpVec3(0,1,0), oldDir); //Rotation Matrix for oldDir
  newMatrix.makeVecRotVec(fpVec3(0,1,0), newDir); //Rotation Matrix for newDir

  //change into Quaternion
  oldMatrix.getOrthoQuat(oldQuat);
  newMatrix.getOrthoQuat(newQuat);

  //create the interpolated quaternion at time t between 0..1
  slerpedQuat.slerp(tslerp, oldQuat, newQuat);
  slerpedMatrix.makeQuat(slerpedQuat);

  newDir = pfVec4(0,1,0,1) * slerpedMatrix;
}
```

3.3 Summary

Travel, the process of moving through the environment from the current position to a single target, is the main interactive task for exploration and one of the major activities in virtual environments. In virtual environments, this is typically done by direct user control of the view, which often requires considerable skill to control the interaction. Also, it is necessary that the position of the target in the virtual environment be known to the user. This may not apply to visually hidden targets or non-visual targets like sound and olfactory sources.

In this section, we introduced a new, real-time capable approach to automated travel in dynamic, unpredictable virtual environments that is based on the potential field method. It is derived from the physics of the motion of a charged particle in an electric potential field. Originally, this local method was used in robotics for obstacle avoidance and fast path planning. The method uses a discretized representation of the environment in a uniform rectangular grid, the cube space.

Automated travel in virtual environments is technically equivalent to automated view or camera motion. In our method, the target attracts the camera, while obstacles repel it. We show that for the generation of the motion vector it is not necessary to generate and add all attractive and repulsive potential fields and then derive the result at the position of the camera. The same result is achieved by pre-deriving the potential field functions and adding the final vectors of the resulting force functions at the position of the camera. This reduces the computation to an addition of forces.

The potential field method is a local method, which does not require planning. By applying this method to dynamic virtual environments by means of the CubicalPath system, we can provide for a real-time system for automated travel which moves the view to a single target while avoiding obstacles. In a dynamic environment, the target and the obstacles may move themselves. For the rare case that the method moves into an unwanted local minimum in cluttered environments, we rely on the informed user to move out of this situation by using the generally available interaction tool for navigation.

The system uses a bi-directional data structure which connects the discretized environment, the cube space, with the objects provided by the virtual environment application. Each element (cube) of the cube space generates an attractive, repulsive, or neutral field. The potential field itself is analytically defined by the values in each of the cubes, which are attributes to the pre-defined force field functions.

The motion through the environment is generated in three stages which are cycled continuously and executed if appropriate. In step one, the visible geometry is discretized and assigned to the cube space (geometric setup). In static environments, this step is skipped after the initial setup. In dynamic environments, this step demands a fast discretization method which will be introduced in Chapter 5. Step two once sets the target (target setup). In step three, a new view is generated depending on the characteristic of the potential field defined in steps one and two. In addition to the core potential field motion, we take care of the orientation of the camera and ensure the quality of motion by controlling the velocity of translation and rotation. The three steps are repeated until the camera (the view) arrives at the target.

4 Guided Exploration using Dynamic Potential Fields

Exploration is the goal-driven activity of moving in an unknown spatial environment for the purpose of discovery. It involves the core motion task and the cognitive task of deciding where to move. Exploration is one of the typical motivations of using virtual environment, as these are designed to present spatial information.

While travel is motion along a path to a well-defined target, exploration either does not specify a target at all or it considers, driven by the goal for exploration, multiple possibly interesting targets. The process of exploration is divided into a cognitive task (wayfinding) and a travel task[BKH97]. The cognitive task is to identify the targets to be visited or at least the immediate direction of motion. A decision has to be made before the travelling may start. People commonly use external information for decision making not only in the spatial domain. A good example is internet search engines. A query, in the form of a question or a list of attributes, results in a list of targets in decreasing order of relevancy. If a queriable information space exists for the virtual environment application, supporting the user in the cognitive task of selecting possible targets, then a connected travel support system must deal with the resulting field of more or less interesting targets in the context of the goal. This field is called the **goal field** (see Section 1.1.2).

The aim of this chapter is to integrate output of existing queriable information spaces into our system and to move the user to the resulting targets for the purpose of providing guided exploration. The input to the system now consists of a field of interesting spatial targets with a value indicating their relevancy in the context of the goal, the goal field.

The potential field algorithm introduced in the last chapter is a real-time capable algorithm for generating the motion of a camera. It moves the camera into a local minimum, which forms the position of a target. This method will fail in the presence of multiple targets, as the camera will stop moving when it arrives in the first minimum. Additionally, as this method works locally, a complicated path, for example like in a maze, cannot be generated.

This chapter introduces the dynamic features of the potential field approach used by the CubicalPath system. It describes the adjustment of the attractive potentials after each motion step to enable the visitation of multiple targets. It discusses the processing of interactive input (objects, targets, camera, interaction tool) from the client application. Finally, it introduces some extensions to enable the specification of a predefined path or an explicit view even though a local method like potential fields is utilized.

4.1 The Dynamic Potential Field Method

The process of generating camera motion with dynamic potential fields extends the method in the last chapter by an additional step which is described below. The process now includes the four steps illustrated in Figure 4.1, which are cycled continuously. The steps are the following:

1. **Geometric Setup:**
   The geometric setup, step one, is equivalent to the setup described in Section 3.2.1.

2. **Goal Field Setup:**
   The second step of the method is the setup of the goal field, the list of targets in the field. Targets are supplied by the client application. A user query in the application is processed utilizing its information space and results in a list of objects which match more or less the query. The values for the targets are received in form of a table which states for each object its level of relevancy to the goal, which is set to be its attractivity. The attractivity value is a floating point number, which allows the specification of different degrees of attractivity (more or less interesting objects).

   The behavior of the camera is controlled by the attraction functions (see Section 3.2.2), as in the automated travel method. The only difference here is that α, the attractivity, can vary. It was fixed to 1 for automated travel.

3. **Step Generation:**
   In step three, the orientation of the camera now has to consider multiple targets. An approach will be discussed in Section 4.4.1. The generation of the next camera position, is equivalent to the motion generation for the automated travel method described in Section 3.2.3.

4. **View Analysis**:
   Step four is required for decreasing the attractivity of visible targets. The reason and the approach is explained in the following section.

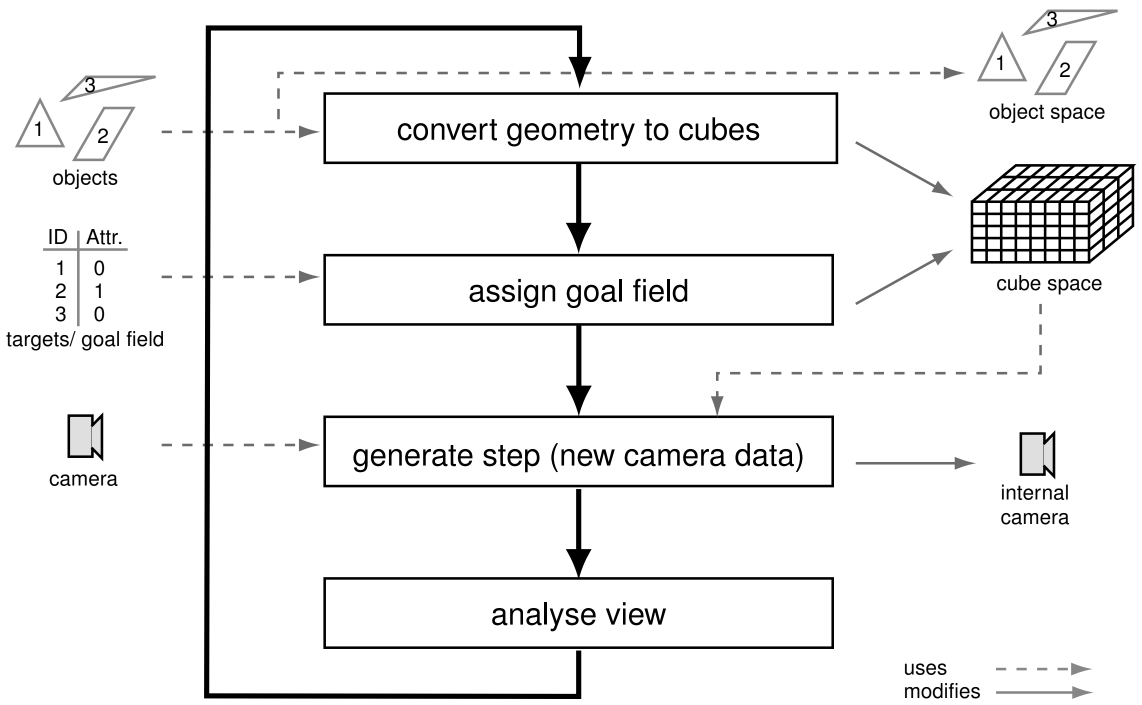


Figure 4.1: Dynamic Potential Field Method. Process diagram.

4.2 Dynamic Attraction Fields

The CubicalPath algorithm works with global and local minima to move a camera. Each target generates a function with a global minimum at the position of the target and contributes at least a local minimum to the compound field, if not a global one. Wherever the camera is inserted into the field, it will move into the next minimum. Even if this minimum was not a target, and there are more targets to visit, the camera will stop moving and by this stop presenting targets.

To overcome this problem, the compound field of the attractive cubes is adjusted in each step. This adjustment is based on the assumption that if a cube, thus an object, is viewed for a while, it loses attractivity to the user. In real environments, users would decide for themselves when they lose interest in an object in front of them, and then walk away. In terms of guided exploration this means that the user should be able to view each target for a certain amount of time in order to understand its impact to the exploration task. After a while, the user will lose interest in this object, and the next target needs to be presented.

The CubicalPath system achieves this by decreasing the level of attractivity of objects in view and thereby adjusting the attraction fields. For this, the adjustment process needs to know which objects are in the view of the user. It can then decrease the attractivity of these objects. Thus, after a while, the local minima of viewed objects flatten out until they do not contribute any more to the compound field. The camera is then ”released” from these targets.

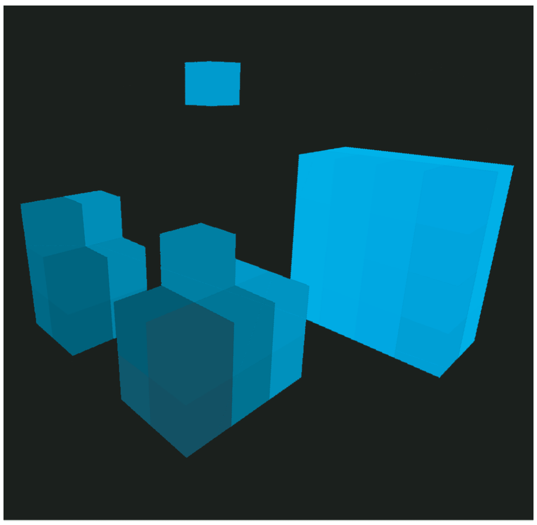


Figure 4.2: CPAnalysisServer. Analysis of cube visibility for a "room" scene. See Figure 6.6 for the original geometric scene viewed from a slightly different angle.

4.2.1 View Analysis

For guided exploration we need a method which decides for the user when they viewed an object long enough. Our method is based on the assumption that the larger a target is in the view, the more likely it is that the user's attention is on this target. We use a method which retrieves the "amount of visibility" for each target. It analyzes how much of each cube is visible in the current view. Distant cubes or partly hidden cubes will be less visible than close, non-occluded cubes.

This is done by a separate server, the *CPAnalysisServer*. This server is implemented independently, because it needs a graphics context. It receives a list of all obstacles (their cubes), the current camera position, and a list of targets. This server is used for

- finding a target which is visible from the current camera position (later used for orienting the camera to a visible goal),
- analyzing how much of each target is seen in the resulting view. This is used for adjusting the appropriate values in the dynamic potential field described in the following section.

This method generates, with the assistance of graphics hardware, a list of visible, attractive cubes by perspectively rendering all attractive and obstacle cubes color-coded in the viewing frustum of the viewer, as in Figure 4.2 for a "room" scenario. The rendering parameters are set to match the rendering of the application view. The color of the cubes identifies the original position of the cube in 3D space by mapping the x, y, z position in the cube space CS on a RGB color (e.g. $R = x/255$; $G = y/255$; $B = z/255$; for $x, y, z \in \mathbb{D}$) or an index color (e.g. $IndexColor = x + y * 255 + z * 255 * 255$; for $x, y, z \in \mathbb{D}$). The resulting image is analyzed, and a histogram of each color is built. Cubes closer to the view position will overlay a larger area of the image than distant cubes. Occluded cubes are only partly visible or not visible at all. Therefore, distance and occlusion will map to the number of pixels of one color that is found in the image. In this method, it is possible to value areas of the view differently. For desktop based systems it can be assumed that the user's attention is more focused on the middle of the screen than on its sides. Thus, when analyzing the image, regions in the middle of the view could be valued larger in terms of user attention than border regions.

Finally, the color is mapped back to the corresponding cube, its position in the cube space, and a list of cubes sorted by its visibility – the number of pixels of this color – is sent back to the CPServer, the server module which implements the CubicalPath system.

4.2.2 Goal Field Adjustment

The CubicalPath system uses the list of visible cubes generated by the CPAnalysisServer for the reduction of attractivity values of a cube. There are two possible modes for adjusting the attractivity. These largely influences the behavior of the camera. First, in the *cube mode*, the values can be adjusted for each cube separately. For an object consisting of multiple cubes this results in the camera moving around the object, until every single cube has lost all of its attraction. A cube loses its attractivity α by the function in Equation 4.1.

$$\alpha_{cube} = \alpha - (\xi_{decrease} \frac{pixels}{winSizeX * winSizeY}) \tag{4.1}$$

The term $\frac{pixels}{winSizeX*winSizeY}$ scales the number of the pixels ($pixels$) in the image belonging to this cube relative to the image size. The image size is defined by its width $winSizeX$ and its height $winSizeY$, both measured in pixels. This term has a maximum value of one[1].

The second mode, the *object mode*, adjusts the complete object, once one of the cubes is viewed. This mode makes the behavior of the camera independent of the size of the objects. In this method, the camera views the object for a period of time, which is independent of the number of cubes, before moving on. Also, the object is only viewed from the direction it is approached, because here, even if multiple cubes represent an object, the camera does not need to move around the object to decrease the attractivity of each cube individually, before the camera is fully released from the object.

Equation 4.2 adjusts the attractivity per object.

$$\alpha_{object} = \alpha - (\xi_{decrease} \frac{\sum_{i=1}^{n} pixels_i}{winSizeX * winSizeY}) \tag{4.2}$$

[1]This is the case, if the real pixel count values in the image are used. If the pixels are valued relative to their position in the view, then this value may be larger

$\sum_{i=1}^{n} pixels_i$ is the sum of all pixels collected for the n cubes comprising this object. The resulting object attraction α_{object} is then assigned to all cubes that represent this object in cube space. The distribution of α_{object} to the cubes is done via the list of cubes lq, defined in Section 3.2.1.

With these *dynamic attraction values* it is ensured that multiple targets are visited sequentially. Furthermore, the attraction value correlates with the duration an object is visited. The larger the value of α, the longer the duration of stay relative to objects with smaller values of α. For example, if the α value of an object which consists of one cube is set to 8 and this object is covering the complete view, then it would take 8 steps to decrease its attractivity to zero. $\xi_{decrease}$ is a global scaling factor which controls the overall duration of stay influencing all targets equally.

4.3 Interactive Input

To allow for a high degree of user control, any decisions of the system can be overridden by the user. The control parameter to the CubicalPath system were illustrated in Figure 4.1. They were the *objects' geometry*, the *targets with their attractivity* and the current *camera position*. All three parameters can be changed at any time during the continuous motion generated by the CubicalPath system. The first two change the potential field immediately, while the third is used in the calculation of the next step. This provides for a fully interactive presentation, which is necessary for the highly interactive task of exploration in virtual environments. For example, objects may be moved during presentation in simulations or collaborative virtual environments. The targets may change because the user re-queries the information space. The users may wish to further explore a specific object by stopping the presentation. They can resume or re-query the CubicalPath system whenever desired. They can navigate themselves or overlay their navigation to the motion generated by the system.

The impact of changing these three parameter during run-time is explained in the following sections.

4.3.1 Dynamic Camera and Interaction Tool Input

The advantage of the potential field approach is that it stores all the geometric and attractivity information independent of the camera data. This allows the client application to change the camera position used for the step calculation at any time without needing to reconfigure or update the system's data.

The camera data independent specification of the setup data can be used to reflect user input or application changes of the camera data into the CubicalPath system during runtime. For example, it is possible to merge the output of the CubicalPath system camera with the input of an *interaction tool* in the client application. This

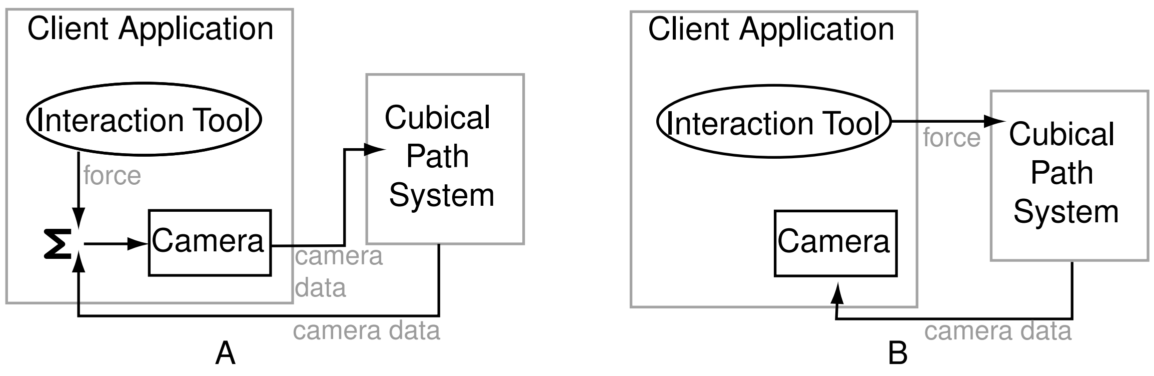


Figure 4.3: Two Ways of Interactively Influencing the Camera Motion. In A, the client application merges the force input of the interaction tool with the new camera data from the CubicalPath system to generate new application camera data. In B, the force input is sent to the CubicalPath system where it is included into the calculation of the new camera data.

enables the user to influence the motion of the view by steering in the desired direction.

There are two ways to integrate the input of an interaction tool into the application. The first, illustrated in Figure 4.3A, is to calculate the overlay in the client application. The CubicalPath system sends its results, the camera data, to the client application and the client then accumulates the output of the interaction tool with the output of the CubicalPath system. The resulting camera data are then sent back to the system to make the modified camera available for the next calculation step. This results in a **dynamic camera input** by the client application to the CubicalPath system.

The second way of interactively influencing the generated motion, illustrated in Figure 4.3B, is that the client application sends the output of the interaction tool to the CubicalPath system. This output is typically a force vector. This results in transmitting an **interaction tool force input** to the CubicalPath system. This force generates a third local field function, which is added to the existing attractive and repulsive fields. This function drags the camera in the direction of the force. If this force is strong, then it may override the results of the CubicalPath system. If it is comparable to the internal forces then subtle changes to the movement of the generated camera are possible. The results of the calculation are sent to the client application and directly manipulate the view. An example is presented in Chapter 7.3.

4.3.2 Dynamic Objects

If objects in the client application are transformed in any way, this transformation has to be communicated to the CubicalPath system and applied to the cube space CS. This requires re-voxelizing the object, removing old object information stored in the corresponding cubes from the CS and adding new information to the new occupied cubes. If this is done, while a user query is mapped into the CS, then the current information for this specific object – its current attractivity – needs to be preserved. This cannot be done per cube, as there is no knowledge of the new position of a specific cube. It may not exist any more. Therefore, the attractivity information is applied on object level.

If the mode of adjustment is in object mode, then the attraction value for all cubes is equal and already stored in the according object. If the mode is the cube mode, then each cubes attractivity α_{q_i} has to be collected, normalized, and assigned to the object. The function used is

$$\alpha_{object} = \frac{\sum_{i=1}^{n} \alpha_{q_i}}{n} \tag{4.3}$$

where n the number of cubes forming the object. The value α_{object} is then assigned to the new set of cubes occupied by the object.

This procedure preserves the field information. It allows, for example, to follow moving targets. The field is continuously modified by moving the area of attractivity, defined by the attractive cubes, through the field.

Dynamic scenes require real-time voxelzation of the dynamic objects. Methods for this are described in Section 5.4 of the following chapter.

4.3.3 Dynamic Targets

If, during the continuous calculation of camera data, new attraction values are set by the client application (for example on basis of a new user query), then these are assigned dynamically to the cube space and change the compound field. The camera will immediately react to these changes and start presenting the new targets from the current camera position. This, in effect, dynamically adjusts the goal field.

4.4 Extensions

4.4.1 Camera Orientation

For automated travel presented in the previous chapter, the camera was oriented in the direction of one exclusive target. In the presence of multiple targets, a decision has to be made, as to which target should be inside the view. If the camera is oriented in the direction of motion of the camera, it can not be guaranteed that the camera views a target. Multiple targets may concurrently attract the camera and the influence of repulsive forces makes the camera avoid obstacles, which usually results in the camera moving to positions in between the targets and not directly to a target.

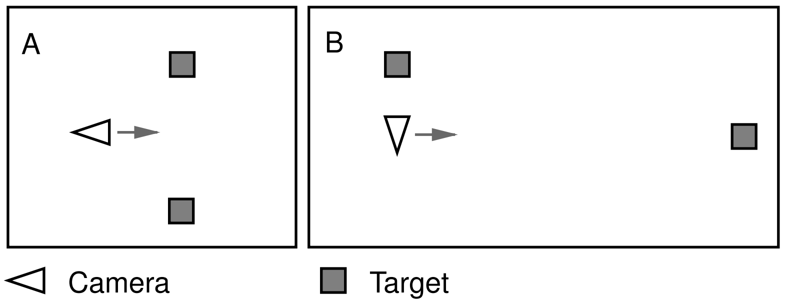


Figure 4.4: Examples for Camera Orientation.

Figure 4.4A shows an example where the camera is attracted by two target at the same time. The camera will eventually stop in between them both but still view in the direction of former motion. In this case, none of the targets is in view. In the case that the camera moves in a direction with no target in close view, a camera view in direction of motion results in images without meaningful content. If, for example, a minorly interesting target is situated in a 90° angle to the direction of motion, it makes sense to present this interesting object. For this, the camera rotation has to turn 90° to move this target in the view, while the forward motion of the position remains unchanged (see Figure 4.4B). Visually, this results in panning of the camera as it passes the interesting target. After passing this target, the camera may turn back to straight ahead viewing.

These examples illustrate that the orientation of the camera has to be considered, in order to generate a presentation. A straight view in the direction of motion is not sufficient for the camera view. When calculating the orientation of the camera, the system decides which cube – thus, which target – should be viewed. It also ensures that the chosen cube is not occluded by other cubes holding geometry.

Algorithm

Algorithm 4.1 first generates a list of all cubes that have an attractivity α larger than zero. The list is sorted by $\frac{\alpha}{d}$ where d is the distance of the camera to the cube. Therefore, the first element in the list is the cube with the highest attractivity scaled by its distance. The cube in this element can be checked for visibility (see Section 4.2.1) and discarded if not visible. The camera will point in the direction of the first visible cube in the list. This results in cubes, thus targets in the view which have a large attractivity or which have a small attractivity but are close to the camera.

Algorithm 4.1: Calculation of the Rotation of the Camera

```
//generate list of attractive cubes valued by their distance
list = generateAttractiveCubesList()

while (check_visibility(list.top(), cameraPosition) is FALSE)
    //discard top element from list
    list.pop()

//point camera to top element
cameraLookAt(list.top())
```

4.4.2 Predefined Paths

As explained before, the CubicalPath algorithm is a local method. The attractive force of one target on the camera is always in the direction of the target. This method has no sense of the environment in between the camera and the target. If there are concave obstacles in the way, like in Figure 4.5*A* and *B*, the camera might move into this obstacle, drawn by the target behind it. There will be no way out, unless the target releases the camera from the obstacle. With convex objects this will not happen as the camera normally slides along the borders of the obstacle. One solution is to allow only scenes with convex objects or to fill the object space of concave objects to make them convex like in Figure 4.5*C*.

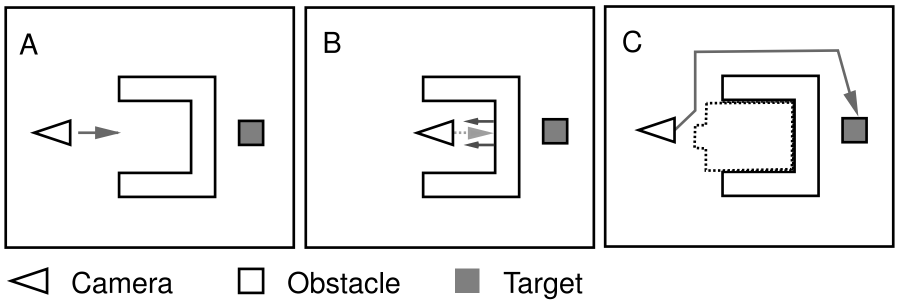


Figure 4.5: Deadlock Situation for Camera.

This does not overcome the issue of large planar obstacles like walls. One example is illustrated in Figure 4.6. In illustration *A* the camera is attracted directly by the target. The camera moves to the wall, which will repulse it. A local approach usually will fail, unless by chance it lines up with a passage way. This limits the area of application for the potential field method to applications consisting of convex, filled objects with enough space for movement in between them.

To overcome this problem, global knowledge can be assigned to the CubicalPath system, such as information about at specific path to an objects. To assign global knowledge to the CubicalPath system, which may exist in the client application, *navigation objects* are introduced. A navigation object is a new, invisible type of object, which can be used as an obstacle to the camera or as an invisible target.

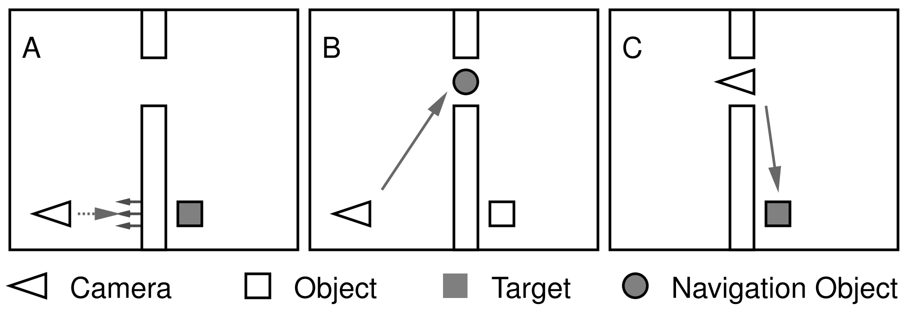


Figure 4.6: Navigation Objects for Specific Path.

They force the camera to use a certain path or to move to a specific location. Navigation objects are activated and have to be reached before the visible target itself is activated. They can be used if specific sub-targets are to be reached along a path where there is no trivial way to the targets or where the pathway is small. In Figure 4.6*B*, the target's attraction is preliminarily turned off while the camera is drawn to the pathway by a navigation object. Once this is reached (see *C*) the navigation object is discarded and the target is set attractive again.

This enables the use of *predefined paths*. They can be implemented by

a) defining a sequence of active targets forming the path

b) passing on of values. Once a cube was visited, it loses its attraction, and at the same time another cube is set attractive. One cube passes its value onto the next one in the list.

Separation of Objects and Targets

A navigation object has different attributes concerning the CubicalPath system than a traditional geometric object. So far, a target always was expected to be a visible object, thus an obstacle. These objects were defined by their geometry and the according cubes in the cube space. A navigation object normally is situated in the free space of the scene. It does not have visual geometry and generates no repulsive force for the camera. It is not connected to any object in the scene of the client application.

This kind of bare positional information also needs to be administered in the system. The CubicalPath system has a notion of geometric objects – controlled by the client application – and discrete cubes – controlled by the CubicalPath system. Both parts are required for vice-versa communication. Thus, this new kind of positional information also has to generate an object. Therefore, the notion of an object has to be extended to include attributes like visibility, obstacle, and target type. This then allows for the definition of targets which are independent of geometric objects of the client scene.

With this extension of the object definition, it is now possible to define navigation objects for the camera independently of the client scene's objects and activate them as needed. An overview of this definition is given Section 4.4.4. Before this, the definition of objects is further extended to allow the specification of a predefined view on an objects.

4.4.3 Predefined Views

The camera will move in a straight line to a target, if not distracted by an obstacle. The final view on the target object – the side which is viewed – is determined by the direction from which the object is approached, as the view is defined by looking from the camera position to the target. For camera control, this might not be sufficient, for instance, if the object has a front and a back, a "nice" and a "boring" side. This calls for a method to define *views* on an object. These allow for director's instructions or provide a certain view onto an object.

Separation of Targets for Position and View

For the definition of a view, a position and orientation of the view is required. The orientation can be represented by the definition of a direction of view vector and the up-vector. In the CubicalPath system, there is no notion of a vector which can be stored in the cube space CS. However, the direction can be derived by the position and a "look at" position, the end point of the direction vector:

$$\overrightarrow{dir} = \overrightarrow{lookat} - \overrightarrow{pos} \tag{4.4}$$

This requires splitting the target information into the specification of a positional target and a "look at" target in the CS. The "look at" target is normally the visible object itself. The positional target is at the desired position of the camera which is in some distance to the target. From this position, it will view in the direction of the "look at" target, the target view. An example is illustrated in Figure 4.8.

With this definition, it is possible to define *position/orientation pairs* for the camera and activate them as needed. This specification is independent of the client scene's objects.

4.4.4 Extended Object Attributes

Section 4.4.2 redefined an object to be a more abstract definition of positional information in the geometric space. An object can be the geometric object in the client application or it can define any position or area, independent of the visible objects of the scene. Section 4.4.3 redefined a target to be a target for the camera position and/or a target for the "look at" location for the view. Both definitions are required to control the potential field algorithm to a maximum extent.

Because of this, there are three attributes added for each object:

- obstacle [bool],
- target position [objectID], and
- target view [objectID].

Every object has at least one of these attributes but may in fact have all of them. These are important attributes for camera instructions and navigation objects. This extends the description of an object in Section 3.2.1 to

- α_{pos}, α_{view} instead of α: the attractivity for the position and the view
- $targetPosObject$, $targetViewObject$: pointers to objects which are set attractive, if this object is set attractive by the client. This may be a pointer to itself, if the target is the object itself. Or it is a pointer to any other object in object space.
- $isSolid$: makes this object an obstacle when generating the position
- $isVisible$: makes this object an obstacle when generating the view

These object attributes are also distributed to the cube space CS. Depending on the pointers in *targetPosObject* and *targetViewObject*, either the α values of the object itself or the values of the objects pointed to are distributed to the CS.

When calculating the new position of the camera, the positional α_{pos} value is used as an attraction value for the camera. All cubes that are set to be obstacles will produce a repulsive force, when the camera gets close.

In the same way, the target view value is applied when the new view of the camera is calculated. Here, obstacle cubes are used to check for visibility. They do not generate a force.

4.4.5 Navigation objects at work

This extended definition of objects will now be demonstrated, with some examples in Figures 4.8 and 4.9. At the same time this section will discuss ways of controlling the CubicalPath method:

All geometric objects are treated as obstacles – like object A in Figure 4.7 – i.e. the camera may not move through objects. The distance the camera is keeping with respect to the object is defined through the repulsive function (Equation 3.21) and its values of γ_{rep} and d_0.

If an object is set to be attractive by the application, an attraction value is assigned to this object and both target attributes are set to a value > 0, as illustrated in

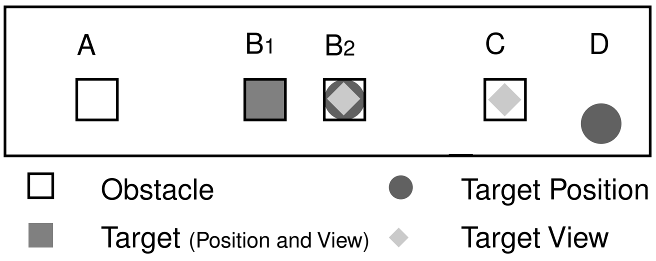


Figure 4.7: Object Attributes.

objects B_1 and B_2. The camera will be attracted by the object and will stop in the vicinity of the object, where the repulsive force and the attractive force of the object become equal. B_1 uses the joint notation, where target is both target position and target direction, while B_2 shows both targets separately. The behavior of both is the same.

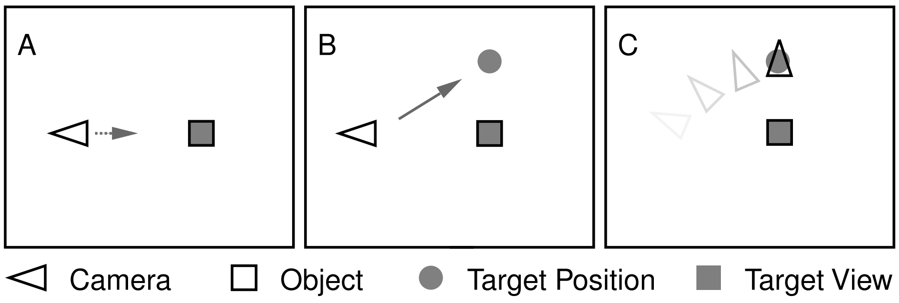


Figure 4.8: Navigation Objects for Specific View.

If a special view of an object C is requested, then a second object D can be introduced. D forms the target for the camera position, while the target for the camera view is still the object C itself. This is illustrated in Figure 4.8. Here, the camera is on the left side of the object. Normally, it would approach the object from the left side, thus view the left side. If the side of the object which should be presented is on top of the object in the 2D illustration, then, with the help of a navigation objects, it would be possible to make the camera view the object from this side.

To summarize the influences of the different object attributes in Figure 4.7, in the case of A the camera avoids the object when it gets close. In the case of B, the camera position can be at any point around the object in the distance defined by the repulsion function. In combined cases of C and D, the camera moves to D and then will view C. As D is not an obstacle, the camera can move into the space occupied by D.

Figure 4.9 shows the different end positions, viewing direction and paths of a camera for the same geometric situation without (case A) and with (case B) the usage of a navigation object for defining a view.

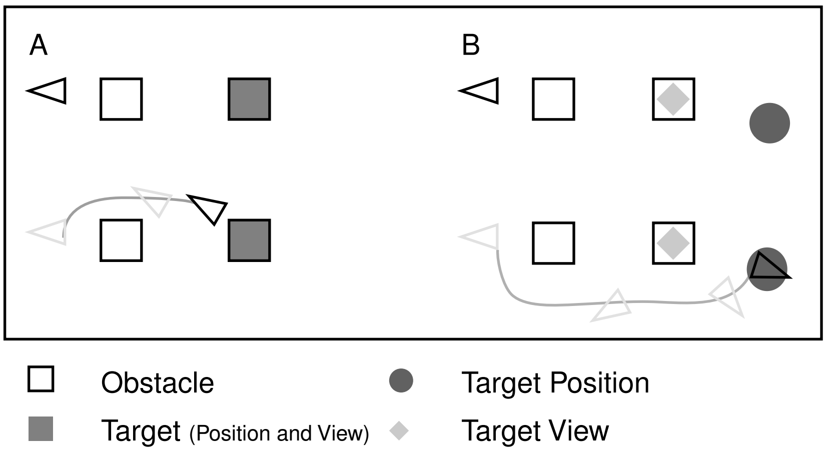


Figure 4.9: Examples of Navigation Objects.

4.5 Summary

In this chapter, we extend the potential field based method for automated travel to an interactive exploration support system. Exploration can be supported both on the cognitive level of decision making as well as on the level of travelling to identified targets. We support the cognitive level by integrating the output of common queriable information spaces into our system. This output consists of a field of targets, which are identified by their name, together with a value indicating their importance in the context of the query. This results in a landscape of more or less interesting targets, called the goal field. We support travelling by automatically moving the user inside the resulting goal field, presenting the identified targets one-by-one. At the same time, we give, if requested, full interactive control over the motion, or we overlay user input with the system output. By this, we achieve the feeling of being in full control of the system though we integrate the possibility of letting the system do the travelling, thus relieving the user from this task and allowing to focus on the exploration goal.

For automatically travelling along multiple targets inside the goal field, we introduce the dynamic potential fields method. This method deals with multiple targets by adjusting the attractivity of targets. The targets are presented one-by-one, ordered according to the attractivity and the distance of the targets from the current view position. The view is constantly evaluated to gradually decrease the attractivity of visible targets. This, in effect, releases the view from visited targets after a while. The target's importance influences the duration of stay in front of the target. By adjusting the attractivity, the history of visited targets is stored in the goal field and the targets will no longer attract the view. The view is oriented in a way that meaningful content – a target – is in the view, if possible.

We extended the CubicalPath system to be able to deal with interactive real-time input by the client application or the user. The input can be *dynamic, unpredictable object locations*, influence on the geometric setup, *dynamic or re-adjusted targets*, influence on the attractivity setup, or a *dynamic camera*, assigning a new current camera view to the motion generation function. It is also possible to influence the results of the motion generation function with an additional force input generated by an *interaction tool*.

By separating the targets for the position of the view and the orientation of the view, we extended the possibilities of presentation in 3D, as the camera orientation is not fixed to the direction of movement. We also introduced the concept of *navigation objects*, additional, often invisible, objects. They allow us to introduce global knowledge into the environment, by enabling the algorithm to follow a complicated path or by making director-specified views possible. Examples are viewing objects from a certain perspective for dramaturgical reasons, or preventing the view from moving into known dead ends. Navigation objects represent spatial, high-level information about the scene and can also be used to specify non-visible targets in virtual environments like auditory, olfactory, or tactile targets.

The final CubicalPath algorithm comprises of four steps, illustrated in Figure 4.1. It extends the automatic travel system in Chapter 3, by including the analysis of the view as step four. Step two is extended, in that now a goal field is provided as input, not a single target, and that in each step, this goal field is adjusted according to the history of visited targets.

5 Real-time Voxelization

This chapter introduces volume data and then describes voxelization methods for converting geometric data interactively into volume data. Volume data will be used by the CubicalPath system as a uniform representation of a spatial environment. Two voxelization algorithms to generate the volume data and their implementations are presented. These are namely a fast software-based triangle voxelizer and a hardware-based voxelization method using standard graphics hardware. The behavior of both methods at different resolutions of the voxel space, different scene sizes and on different platforms are evaluated and discussed.

5.1 Volume data

5.1.1 Definition

Volumetric data or volume data is a set S of samples (x, y, z, v) that store a value v for a coordinate (x, y, z) in 3D space or volume space. The value v can be a single value or a complex data structure like graphical information. If the value v is 0 or 1, then the data is referred to as *binary data*.

A volume data set can consist of data sampled at random locations or it can be structured in some sort of grid. In many cases, the sample locations lie in a regular grid and samples are taken at regular constant intervals along the three orthogonal axis. A set is called *isotropic*, if the intervals in all three dimensions are equal, or *anisotropic*, if the constant interval size differs for each of the three axis. Data in regular grids is normally stored in a 3D array which is sometimes called *volume buffer*, *cubic frame buffer*, or *3D raster*.

5.1.2 Example Applications

Examples of anisotropic regular volume datasets are data that is generated by computer tomography (CT) or magnetic resonance imaging (MRI). In these methods, a scanner collects density values for each 3D point in the grid.

Computer Tomography yields series of X-ray-based cross-sectional images of solid objects. CT scanning is used, for example, in medical diagnostics and for non-destructive analysis of the interior of composite material. Magnetic resonance imaging is based on the principles of nuclear magnetic resonance, a spectroscopic technique used by scientists to obtain microscopic chemical and physical information about molecules. Both methods generate volume data by combining series of consecutive cross-sectional images.

An example of a mixed dataset with fixed x and y resolution and unstructured z resolution is geoseismic data. They are generated by igniting small explosions on the ground and measuring acoustic reflective properties of the ground. The sound is reflected by layers in the ground, where the density of the matter changes. The resulting data of this method consists of the amplitude values for a computed depth – using the time – at fixed x and y positions.

Visualization

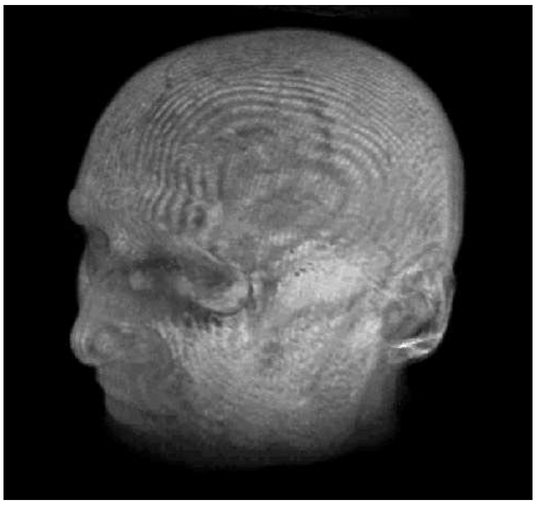


Figure 5.1: MRI Dataset.

Information in volume data can be visualized with different volume rendering algorithms which are thoroughly discussed in Chen et al. [CKY00]. Figure 5.1 shows an example visualization of a MRI dataset where the shading is based on density values of the tissue. Figure 5.2 shows a seismic dataset in a geoscientific application. Here the information is visualized by color coding intensity values in the data.

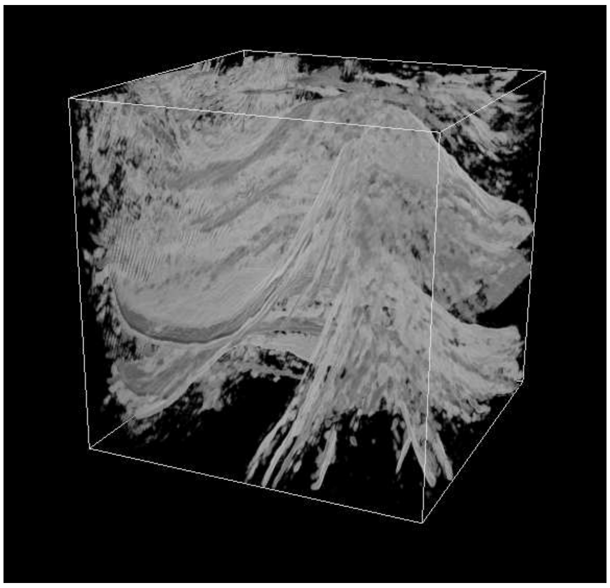


Figure 5.2: Geoseismic Data.

Simplification

Volume data representations can simplify complex geometric data. Rendering the polygonal version of complex models like the model of the Happy Buddha (Stanford University Computer Graphics Laboratory) can be time consuming. The Happy Buddha model, which is visualized in Figure 5.3A, consists of 543.652 vertices and 1.087.716 triangles. In tests on an SGI Onyx 2 the rendering time was $0.5s$. For non-static scenes this results in a visually disturbingly slow image refresh rate. Especially interactions at a framerate of 2 fps are annoying.

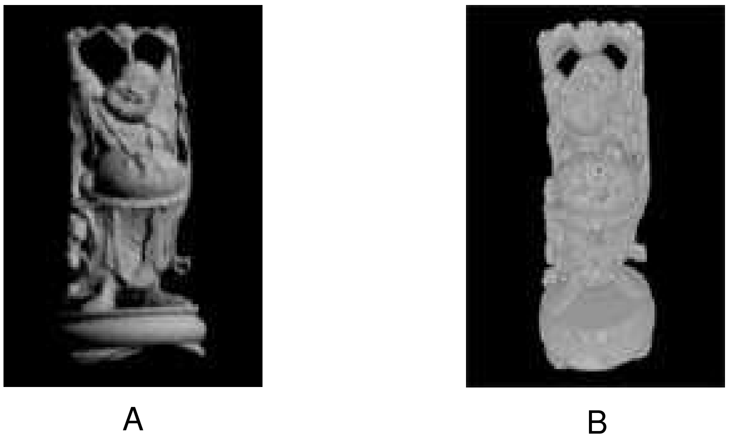


Figure 5.3: Happy Buddha: Polygonal Representation and Volume Data Representation.

The Happy Buddha model in a 255^3 volume data resolution as in Figure 5.3B can be visualized on the same machine with a framerate of 50 fps. This results in smooth visualization and interaction. Here, volume data can provide for a simplified representation of geometric data while preserving the visual attributes and the shape which is needed for interacting with a model.

Spatial Analysis

The above mentioned two examples use volume data to produce visual output. The second example, one possible solution to rendering the Happy Buddha at interactive speed, uses voxelization, the mapping of the geometric data into the volume space. This simplifies the rendering of the complex geometric data to a small subset, the rendering of 256^3 voxels. Volume data representation of geometric models can also be used for simplifying computations on the data, that otherwise would be complex due to their size or due to the spatially opaque way polygons are stored in a scene.

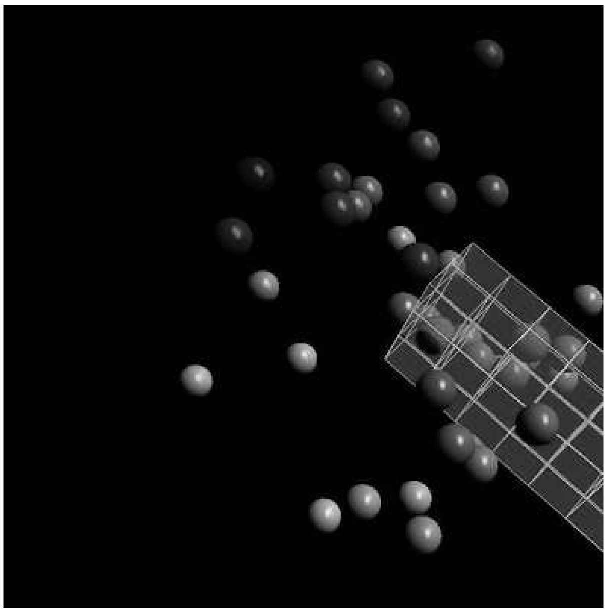


Figure 5.4: 3D Spatial Cursor.

For example, voxelization can be used to analyze the subspace of a 3D Cursor, for the purpose of identifying objects that are completely or partially inside the cursors volume. Figure 5.4 shows a scene consisting of spheres with a rectangular box as the cursor. One common solution to find the intersection of the cursor volume with the objects is to test all the objects geometry against the cursor's geometry. Another approach is to voxelize the scene in the subspace of the cursor's geometry in a suitable resolution and to analyze the resulting volume. Instead of shading of the volume data – like in the Happy Buddha example – the color of a voxel identifies an object by its ID. The advantage of this method is that, additionally to a binary inside/not inside information, it is also known, which specific object is at at which position in the cursor's volume.

Real-Time Camera Motion

Another example of applications requiring 3D spatial analysis is our method to real-time camera motion. Camera motion – we need it for automated travel and guided exploration in the CubicalPath system– is concerned with bringing specific objects into the view, by moving the camera in the scene. For camera motion abstract information like "these are obstacles" and "these are targets" are connected to the objects of the scene. This requires spatial analysis of the complete scene. Mapping the analysis data into the volumetric domain results in a simplified, uniform, and more abstract description of the scene. This can then simplify and speed-up subsequent calculations.

Camera motion has to be generated real-time and can not be pre-computed when there are dynamic objects, autonomous agents or interactively modified objects in the scene. The voxelization methods described in this chapter will be utilized to do real-time camera motion. The requirements concerning the volume data generation and representation for real-time camera motion are listed in the following section.

5.1.3 Voxelization Requirements

The CubicalPath system requires a volume data representation of the geometric scene for the geometric setup of the cube space. For this task, the voxelization process should meet the following requirements:

- **generation of object $\leftrightarrow$ voxel information**: information about which objects occupy which voxel. This is required to address the corresponding voxels when information in the objects change and vice versa.
- **fast (re-)voxelization**: up-to-date representation of dynamic objects, objects that are transformed during the application.

The next section discusses some common voxelization methods. In this chapter, the term voxel is used to denote a volume element in the volume data set. The CubicalPath system later assigns the voxel information to the according cube information in the cube space. The size and the elements of the volume data set, containing voxels, and the cube space, containing cubes, are equal.

5.2 Voxelization

The first voxelization algorithms were *binary techniques* [Kau87a, Kau87b]. They are an extension of the 2D scan conversion methods – which, for a given 3D model and viewport, identify the final pixels in the frame buffer – into the third dimension. Point sampling evaluates a continuous object at the voxel center and the resolution of the 3D grid determines the precision of the discrete model.

Binary voxelization techniques generate topological and geometrically consistent object representations but they suffer from object aliasing, when they are rendered. Two techniques were developed to overcome this problem, namely filtering and distance field techniques.

Filtering techniques low-pass filter the objects [SK99, WK93]. They smoothly blend the inside and outside of the voxel. Filtered voxelization couples voxelization and visualisation. Compatible filters are used in both steps.

Distance field techniques [Gib98] [Jon96] compute the linear or unbounded distance to planes, edges and vertices and set the density according to the minimal distance to the geometry.

For real-time camera motion, here especially for spatial analysis, the volume rendering stage does not apply and therefore the visual drawbacks of aliasing are not a concern. Also, object presentations do not require a precise outline of the objects as a camera or user viewpoint usually tends to keep distance to the objects. Therefore, it is sufficient for real-time camera motion to use binary voxelization techniques. One solution to this is presented in the following Section 5.3.

For complex models the voxelization time for these techniques is still far from being interactive. These techniques are sufficient for any application where the voxelization can be done in advance. When the application contains dynamic models like in dynamic and interactive environments the model has to be re-voxelized whenever modified. This requires a fast solution to voxelization.

Graphics hardware is utilized by Fang and Chen[FC00] to voxelize CSG trees after they have been converted to triangle meshes. This hardware-based approach to voxelization uses standard graphics hardware to accelerate the rasterization of objects [CKY00]. It preserves the visual attributes of the objects. Fang and Chen use the results for visualization. Their algorithm generates slices of the object model using a surface graphics processor to form the final volume representation. Depending on the graphics processor and resolution of the voxel space this allows fast interactive voxelization. Their approach generates a volume representation of the model which can then be rendered by 3D texture mapping.

Section 5.4 introduces an approach to voxelization for spatial analysis, a non-graphical application to voxelization, which also utilizes graphics hardware.

Voxelization process

The *process of voxelization* for the CubicalPath system is divided into three steps:

- *collection* of geometry
- *voxelization* – mapping geometric data into the volume domain

- *assignment* of voxelization results and additional information like objects references to the volume representation, the cube space, used by the CubicalPath system.

The following two voxelization methods first present the algorithm and then the implementation of the process using the application database.

5.3 Software-based Triangle Voxelization

Voxelization or rasterization means *mapping* the polygonal representation of geometry into the 3D grid domain by preserving the spatial information. A polygon is an infinitesimal thin face in 3D space bound by the polygons edges. Voxelization is required to identify positions in the 3D grid which are sliced by the face. Additional attributes of the polygons, for example IDs or visual properties, also require mapping to the 3D grid. For IDs this means assigning the polygon's ID to each of the sliced locations in the 3D grid.

Triangles are the most simple and basic 3D object representation and all other polygonal representations can easily be decomposed into a set of triangles. Thus, with a triangle voxelization method, all other polygonal representations also can be mapped into the volume domain.

For the purpose of triangle voxelization a 3D scan conversion based triangle voxelizer was developed. It extends the 3D Line Voxelization Algorithm by Cohen [Coh94], which guarantees to visit exactly all the voxels along a 3D continuous line. Section 5.3.1 describes how the triangles are voxelized using the Triangle Voxelizer. The following section describes the setup of the application scene-graph to identify specific objects in an arbitrary geometry file, to collect the triangles from the scene-graph and to initialize their voxelization.

5.3.1 Triangle Voxelization Mechanism

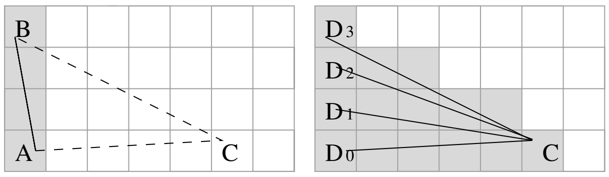


Figure 5.5: Triangle Voxelizer. Voxelization of a triangle with the Triangle Voxelizer.

The voxelization algorithm proposed by Cohen [Coh94] generates a sequence of all voxels visited by a 3D Line with endpoints on an regular integer grid, a grid with intervals of size one. Each voxel along the 3D line is face-adjacent to its predecessors. By determining the face from which the continuous line departs one voxel, the next pierced voxel can be identified.

Now consider a triangle with edges A, B and C, illustrated in Figure 5.5. From the centers D_i of all the cubes that were found on the line between A and B, a line is constructed to C and analyzed in the same way. By this the triangle is filled with a set of lines. At their start point these lines $\overrightarrow{DC}$ are only one cube apart and they converge to each other in direction of C. Therefore, it is guaranteed that with the above algorithm, all cubes crossing the area of the triangle are pierced by a line, thus found. With $n + 1$ cubes visited along $\overrightarrow{AB}$, D_0 and D_n are set to be the exact points A and B. The C++ implementation code of this voxelization method is given in Appendix B.1.

The next section describes, how the triangles are collected out of a scene-graph and how the scene-graph automatically triggers voxelization.

5.3.2 Scene-graph Setup for Triangle Collection

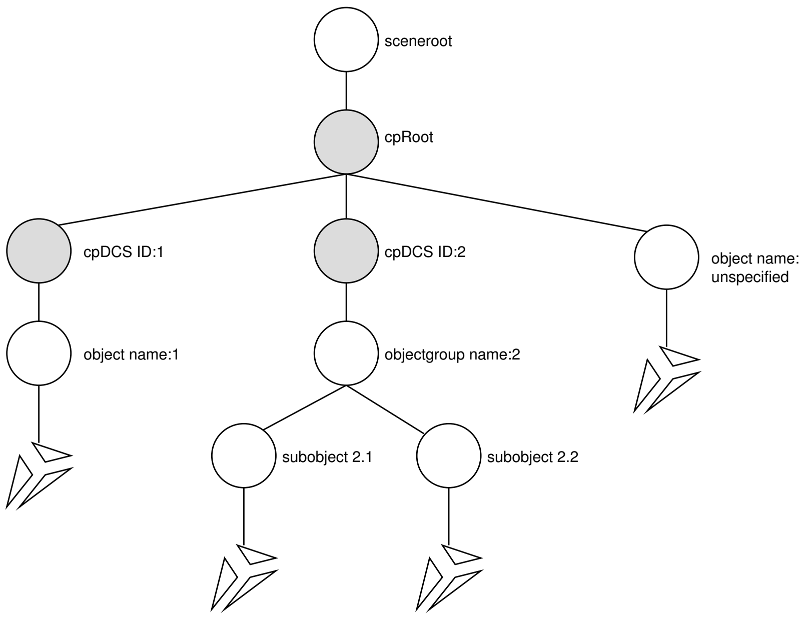


Figure 5.6: Collection of Triangles. Scene graph with nodes supporting the collection of the geometry of objects named 1 and 2.

The database that holds visual and control information in real-time rendering APIs like Performer or Inventor is normally arranged in a *scene-graph*, a hierarchical arranged set of nodes like in Figure 5.6. *Nodes* can hold transformation matrices, geometry information, visual information and rendering attributes. They can also be assigned unique names and any further information. *Relations* between nodes are introduced by the structure of the graph. For example, if objects are moved, the corresponding transformation matrix in the node will change. This will affect the position of all geometry that is arranged in the scene-graph below the modified node. The collection of the data in the database is done by a *traverser*. In each frame, the traverser traverses the scene-graph to collect the current transformations of objects, the geometry and additional attributes for the rendering of the next image. It sets the states of the graphics system depending on the collected information.

The application database, containing the scene to be voxelized, uses the Performer [RH94] scene-graph, because AVANGO is based on Performer (see Section E.4). Geometry in Performer can have different representations like polygons, triangles, triangle strips and quads. To utilize the Triangle Voxelizer first all this geometry has to be collected and transformed into triangles. Secondly, all single objects need to be identified. Therefore, an ID is assigned to each object of interest and sent to the Triangle Voxelizer together with its geometry. Third, during runtime, any changes in geometry have to trigger the re-voxelization process for this specific object.

For applications using Performer these mechanisms are implemented by inserting custom nodes into the scene-graph at the root of each object.

Tasks of Object Nodes

The object nodes – the cpDCS nodes –, inserted at the root of each object

- keep *communication information* to the CubicalPath system.[1]
- store the assigned *object IDs*,
- *collect* all *geometry* in all nodes below this object node,

If the voxelization is performed by the CubicalPath system they

- *send* all geometry as *triangles* to the CubicalPath system with identifying object ID
- *send* the current transformation *matrix*, when the object is moved.[2]

The voxelization can also be performed locally by the application. Then the cpDCS nodes

- *revoxelizes* all *geometry* using the local Triangle Voxelizer implementation
- *send* the *resulting voxel* with identifying ID to the CubicalPath system
- *re-voxelize* and *resend* resulting voxels to the CubicalPath system, when the object is transformed, thus the matrix in the node changes.

[1]Communication between the nodes and the CubicalPath system is done in CORBA. CORBA is a protocol and infrastructure which enables communication between different applications on different platforms. A more detailed description can be found in Section 6. For now it is sufficient to know that there is a communication protocol that can link the nodes in the scene-graph with other applications. This provides for the possibility that the CubicalPath system is a stand-alone server executed on a different system.

[2]This can be utilized to transform the complete set of triangles already residing on the real-time camera motion system side to the new position, orientation and scaling

The root node – the cpRoot – employs a similar behavior as the cpDCS node. It has a fixed default ID. When it traverses the scene-graph, it ignores geometry below a cpDCS node, thus collecting all geometry, that does not reside under a cpDCS. This node ensures that all geometry of a scene is voxelized, even though not all objects are explicitely identified by a cpDCS node.

Inserting Object Nodes in Scene-Graph

A specific object is identified by its name in the database. This name is normally already assigned when the object is modelled. A list of object names together with the root node of the scene-graph (scene-root) is sent to a function which traverses the scene graph, searching for the names in the list and inserts above the root node of each object a new cpDCS node. This is illustrated in Figure 5.6.[3]

Triangle Collection

The collection of triangles is started by triggering an update function in all cpDCS nodes. Each node traverses all its sub-nodes to collect their geometry. If the geometry is not already represented as triangles, it has to be transformed into triangles. After this stage each node sends the set of triangles and its ID either to the CubicalPath system or directly to the Triangle Voxelizer, as described in Section 5.3.2.

If the matrix of a node changes then this node automatically triggers the update process, thus the re-voxelization of the object.

Section 5.6 shows and discusses voxelization timing results of the Triangle Voxelizer. They are compared to a hardware-based approach to voxelization which is discussed in the following section.

5.4 Hardware-based Voxelization

In static environments the voxelization process is performed once and can usually be done in a pre-computation step. Dynamic interactive environments require immediate mapping to the volume space when modified. If complex models have to be voxelized/rasterized, software-based voxelization techniques may become too slow to provide interactive response as they have to rasterize each polygon on the CPU. This operation – rasterization of polygons – is one of the basic operations of graphics engines when generating an image from the object and view specifications. Therefore, it is efficiently supported by modern graphics hardware.

[3]This function is implemented in a special scheme node, the cpDCSReplacer node, which is dedicated uniquely to this replacement task. Applications in AVANGO implement all their behavior in scene-graph structures coded in scheme.

The following sections describe an approach to voxelization that utilizes the graphics hardware. The method identifies unique, single objects by color coding and slice-by-slice rendering of the volume space. The basic approach is extended to identify multiple objects per voxel and to render only dynamic objects.

Section 5.4.1 introduces the rendering pipeline, to show which kind of operations are supported by the graphics hardware. A specific attention is set to the rasterization subsystem. In Section 5.4.2 an algorithm is introduced which utilizes the graphics hardware to accelerate voxelization. The required object identification method is presented in Section 5.4.3. Section 5.4.4 extends the approach to deal with multiple object identification in a single voxel. The construction of the scene-graph to integrate the voxelization process into the normal rendering database is described in Section 5.4.5. In Section 5.5 implementation issues for the used graphics hardware are discussed.

5.4.1 Rendering Pipeline

The process of rendering a polygonal object involves several steps – transformations, clipping, texturing, shading and rasterization – before a pixel is drawn to the screen. They form the *rendering pipeline* that is executed on the CPU and the graphics hardware. OpenGL, the Open Graphics Library, is a widely used software interface to the graphics platform used which consists of the CPU and the graphics hardware. It is a description of a *state machine* for efficient rendering. Depending on the graphics hardware, OpenGL divides the execution of commands between the CPU and the existing hardware components. If, for example, hardware exists only for the frame buffer, all but the per-fragment operations must be implemented on the CPU. Current high-end graphics hardware like the SGI InifiteReality$^{\text{TM}}$and recent PC graphics cards implement most operations of the rendering pipeline in hardware. The hardware units are namely fast *transformation units*, *rasterization units* and the *frame buffer*. Usually, the hardware is designed to directly implement the OpenGL state machine.

The rendering pipeline

Before polygons are sent to the rendering pipeline they have to be extracted from the application database. This pre-step, the *database traversal*, is the traversal of the scene-graph to collect all geometry, attributes and state information that will be rendered or used in the next frame. The results, a set of polygons, lines, points, attributes, and states, will then be sent to the rendering pipeline. The database traversal is dependent on the scene-graph API like Inventor or Performer, and it is always executed on the CPU.

The rendering pipeline (see Figure 5.7) is divided into two major parts:

- the geometry subsystem which deals with geometric transformations
- the raster subsystem which rasterizes the geometry and does pixel operations

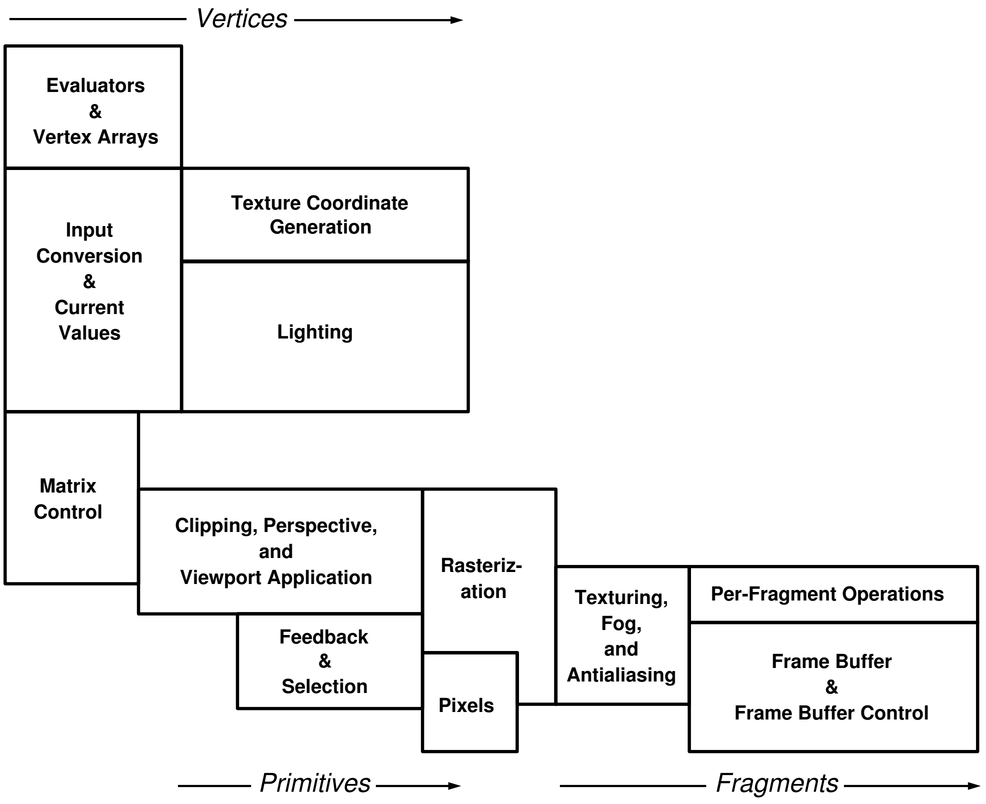

Vertices
Evaluators & Vertex Arrays
Input Conversion & Current Values
Texture Coordinate Generation
Lighting
Matrix Control
Clipping, Perspective, and Viewport Application
Feedback & Selection
Rasteriz-ation
Pixels
Texturing, Fog, and Antialiasing
Per-Fragment Operations
Frame Buffer & Frame Buffer Control
Primitives
Fragments

Figure 5.7: OpenGL Operations. From the OpenGL 1.1 State Machine Diagram [WNDO99].

The *geometry subsystem* operates on the collected polygons, lines and points. OpenGL provides for a flag which specifies, if a polygon is rendered filled, as lines, or if only the vertices are rendered as points. The operations on the primitives are [FvDFH90]: modelling transformation of the vertices and normals from object-space to world space, trivial accept or reject of polygons by testing against viewing volume (culling), per-vertex lighting calculations, viewing projection, clipping of polygons or parts of polygons outside the viewing frustum, and mapping to screen coordinates. These operations are done per vertex and the graphics system implements the required floating point matrix calculations in hardware, usually in parallel.

The transformed primitives then are rasterized and further processed to generate the final pixels of the image. This is done in the raster subsystem.

The *raster subsystem* executes three steps

- the *rasterization* step which rasterizes polygons into sub-pixels that are smaller than the final pixels. In this step also interpolation coefficients are stored.

- a step that applies *fog* and does *texturing* and *antialiasing*. These operations work on the sub-pixels and use the interpolation coefficients. The result of this step are pixel sized fragments. A *fragment* holds color and additional information like depth, stencil and accumulation fragments.

- the *per-fragment operations* which, if enabled, decide, if and how a fragment is applied to the frame buffer. The result is the final color of the pixels on the screen.

The last step, the per-fragment operations, consist of a set of *tests* and *operations* that work on different parts of the *frame buffer*. The frame buffer is a set of pixels arranged as a two-dimensional array. The frame buffer can consist of four types of buffers: *color* buffers (front, back, auxiliary), *stencil* buffer, *depth* buffer and *accumulation* buffer. The front color buffer eventually holds the final pixel color which is displayed on the screen. All buffers have the same pixel resolution but may have different bit sizes per pixel or may not exist at all. Depending on the frame buffer implementation, a system can divide available memory to the buffers. This is done by requesting a suitable *visual* that handles the required buffers and buffer depths.

The tests and operations that are executed on each fragment as it moves through the pipeline are controlled by the OpenGL states. This is illustrated in Figure 5.7. The result of these tests and operations depend on the incoming fragment and the pixel information already in the frame buffer. The results then are applied to the frame buffer itself. For example, a stencil test executes a test on the stencil buffer part of the frame buffer. Depending on the result it may or may not discard the fragment from further processing. The same test can write results into the stencil buffer thus changing the state of the pipeline for following fragments.

The steps in this last part of the rendering pipeline – operations that occur as individual fragments are sent to the frame buffer – are described below and illustrated in Figure 5.8. All tests can be enabled or disabled and some stages allow specific functions to be set by OpenGL.

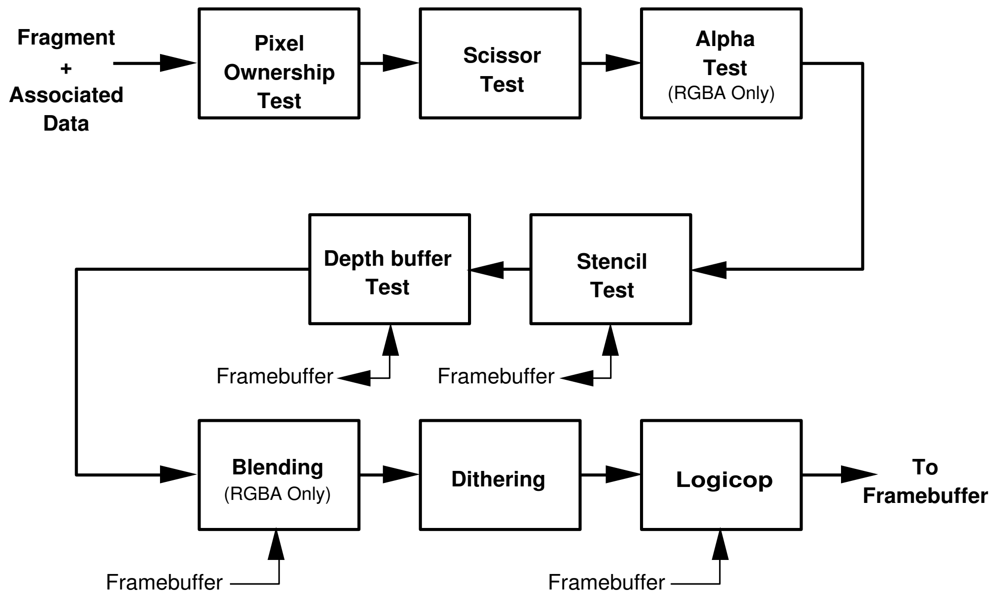


Figure 5.8: OpenGL Per-Fragment Operations. From the OpenGL Specifications 1.2.1 [SA].

Scissor test The scissor test checks, if the fragment lies in a specified rectangular region

Alpha test The alpha test discards a fragment conditional on the outcome of a comparison between the incoming fragment's alpha value and a constant value. The alpha function `AlphaFunc(enum func,clampf ref)` (clampf: floating-point value clamped to [0; 1]) specifies one function out of a set of possible comparison functions.

Stencil test The stencil test conditionally discards a fragment based on the outcome of a comparison between the value in the stencil buffer and a reference value. The stencil function `StencilFunc(enum func,int ref,uint mask )` specifies one function out of a set of possible comparison functions.

The stencil operation `StencilOp(enum sfail,enum dpfail,enum dppass )` specifies what happens to the stencil buffer, depending on the outcome of the test and the subsequent depth test. The operations can be keeping the current value, setting it to zero, replacing it with a reference value, incrementing it, decrementing it, or bitwise inverting it. This test can be useful for multiple-pass algorithms and to restrict drawing to certain portions of the screen.

Depth Buffer Test The depth buffer test discards the incoming fragment if a depth comparison fails. The depth function `DepthFunc(enum func)` specifies one comparison function out of a set of possible functions.

Blending Blending combines the incoming fragment's R, G, B, and A values with the R, G, B, and A values stored in the frame buffer at the incoming fragment's location. It is dependent on the incoming fragment's alpha value, that of the corresponding currently stored pixel and the blending color, equation and factors specified.

The blending equation `BlendEquation(enum mode)` is used to specify one blending mode out of a set of modes. For example, the default mode, `FUNC_ADD`, defines the blending equation as $C = C_s S + C_d D$ where C_s and C_d are the source and destination colors, and S and D are quadruplets of weighting factors as specified by the blending function. The blending function `BlendFunc(enum src,enum dst)` indicates how to compute the source blending factor S and the destination factor D. For example, the blending factor `ONE` assigned to *src* leaves the incoming value as it is and a blending factor `DST_COLOR` assigned to *dst* multiplies the destination color with the blending color C_c. The blending color C_c can be specified with `BlendColor(clampf red,clampf green,clampf blue,clampf alpha)`. Blending computations are treated as if carried out as floating point computations.

Dithering Dithering maps a high-precision pixel color to a color that can be displayed with the current color depth.

Logic Operations A logical operation is applied between the incoming fragment's color or index values and the color or index values stored at the corresponding location in the frame buffer. The function `LogicOp(enum op)` specifies one comparison function out of a set of possible functions.

There are also operations that control or affect the whole frame buffer. They are used, to *clear* specific buffers, to set the *default values for the clear*, to set *masks* which specify which bits are written into the buffer, and to specify the *color buffer* into which color values are written.

The color, stencil and depth buffer were discussed above. The fourth buffer, the *accumulation buffer*, is a color buffer with pixel arithmetic. It holds RGBA color data like the color buffers. The rendering pipeline cannot directly write fragments into this buffer. The accumulation buffer can be used to add the image in the current color buffer to the one in the accumulation buffer. Or it can multiply or add each pixel in the buffer to a value. This buffer is typically used for accumulating a series of images into a final, composite image. With this method, it is possible to perform operations like scene antialiasing by supersampling an image and then averaging the samples to produce the values that are finally painted into the pixels of the color buffers.

Applications of hardware accelerated numerical calculations

The following examples briefly illustrate possible usage of the graphics card to accelerate numerical calculation – calculations that are not primarily used to produce graphical output.

The generation of form factor for the hemicube radiosity algorithm can be accelerated by graphics hardware [BW90, CW93]. A view for each of the faces of the hemicube is specified and the elements are rendered using the rasterization capabilities of the graphics hardware and the depth-buffer for visibility determination. Instead of a color, an item ID encoded into the color information is stored at each pixel, which identifies the visible element.

Ritter et al.[RBD$^+$99] use the accumulation buffer for generating interference between complex wave patterns for computer-generated holograms.

Westermann and Ertl [WE98] use graphics hardware in volume rendering applications. They take advantage of the rasterization functionality, such as color interpolation, texture mapping, color manipulations in the pixel transfer path, fragment and stencil tests and blending operations.

Lengyel et al.[LRDG90] used standard graphics hardware to rasterize configuration space obstacles into a series of bitmap slices. They used the polygon-filled mode to differentiate between obstacles and free space.

Fang and Chen[FC00] utilize graphics hardware to voxelize CSG trees after they have been converted to triangle meshes. Their algorithm generates slices of the object model using a surface graphics processor to form the final volume representation. Depending on the graphics processor and resolution of the voxel space this allows fast interactive voxelization. They additionally rendered polygons in wireframe mode to prevent holes in the model that occur, if polygons are viewed from the side. Their approach generates a volume representation of the model which can be rendered by 3D texture mapping.

The following section will present a voxelization method that can be used for spatial analysis. It will extend the basic voxelization method presented by Fang and Chen[FC00] and will make use of the rasterization hardware, the stencil test and blending and will modify the rendering attributes in the scene's database on-the-fly.

5.4.2 Hardware-based Voxelization and Analysis Mechanism

For the hardware-based voxelization the clipping capabilities of the graphics engine are utilized [FC00]. To voxelize the scene in a given volume data resolution of x, y, z, the scene is rendered orthographically slice by slice, thereby producing z slices of resolution $x * y$. The near and far clipping planes of the viewport are set to bound each slice to be as wide as the current z voxel. Object IDs are encoded into the scene's color attributes, which is explained in Section 5.4.3.

Algorithm 5.1: Hardware-Based Voxelization: resolution x, y, z

```
CreateViewport(x, y)
memory = AllocateMemory(x * y * z)
DisableTexturing()
DisableAntiAliasing()
DisableTransparency()
boundingBox = ComputeBoundingBox(Scene)
// unit cube : (0, 0, 0) to (1, 1, 1)
scaledScene = ScaleToUnitCubeSize(Scene, boundingBox)

zplane1 = 0 step = 1 / z;

for i = 0 to z - 1
    zplane2 = zplane1 + step
    DefineOrthogonalViewingFrustum( zplane1, zplane2)
    ClearFramebuffer()
    DisplayFilled(scaledScene)
        RenderObjectColor(RGB=ConvertIDToRGB(ID))
    DisplayWireframe(scaledScene)
        RenderObjectColor(RGB=ConvertIDToRGB(ID))
    WriteFramebufferToMem(memory + x * y * i)
    zplane1 = zplane2

SendVoxelsToAnalyzer(memory)
```

The hardware clips the geometry outside the current slice – the red slice in Figure 5.9 – and an image (see Figures 5.10) of the geometry in the resulting slice is rendered. The result is written into memory. If the scene is rendered with the visual attributes of objects, like materials, textures and colors, then the results form the volume data representation of the geometric model. This can be used in volume rendering applications or it can be directly rendered as a texture.

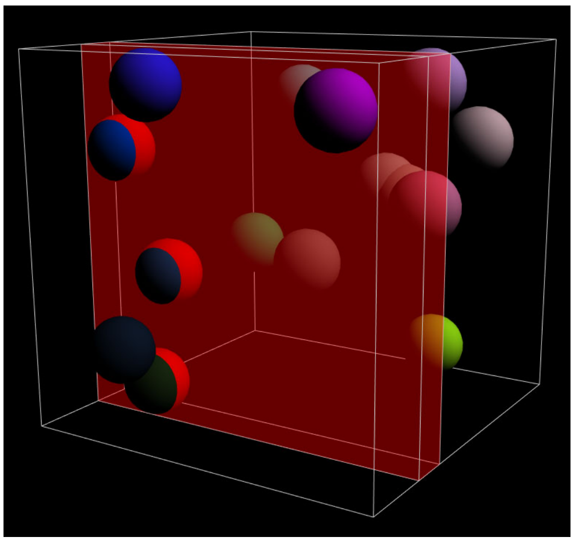


Figure 5.9: Volume Data Generation. One slice of size $x \times y$ is rendered for each z in a volume data resolution of x, y, z

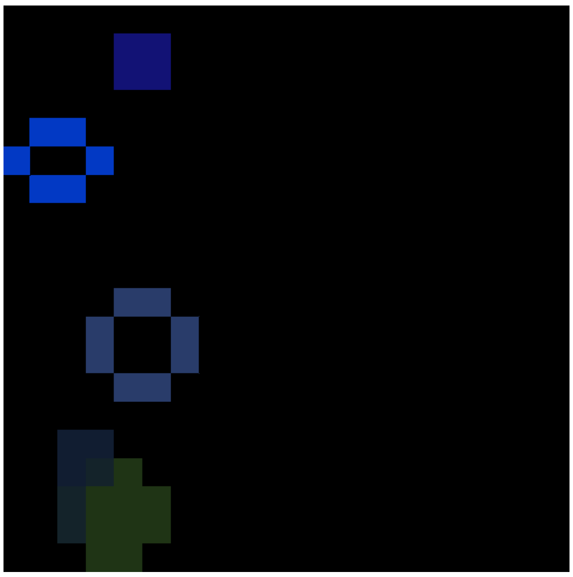


Figure 5.10: Image/Slice in Volume Data. Slice in Figure 5.9 rendered with color-coded objects in a resolution of $20 \times 20 \times 20$. The slice's thickness is 1.

For the purpose of voxelization the colors in memory, which hold the ID information, are analyzed, obtaining for each position of the voxel space the information which object occupies which voxel.

The complete voxelization algorithm is illustrated in Algorithm 5.1.

5.4.3 Color Coding of Object IDs

For the process of identification which objects reside in which voxels, the visual properties of a model are exchanged against information about the objects themselves, namely their ID. The ID is mapped to an RGB color, which is then applied as an emissive material to the object's geometry. All other materials and textures for this object are disabled to ensure that the object is rendered only in the color of the ID. A typical framebuffer has 8 bits per color and thus allows the coding of 2^{24} IDs in the RGB color. The color assignment procedure is illustrated in Color-Coding Algorithm 5.1.

Color-Coding Algorithm 5.1: Identification of 2^{24} Objects in the Volume Space

```
RGB = ConvertIDToRGB(ID); ID ={1,2,..,2^24}
//e.g ID=5 -> R:0 G:0 B:0000 0000 0101

emissive = RGB
ambient = diffuse = specular = (0, 0, 0)
```

The algorithm described in Section 5.4.2, combined with the color-coding approach described in this section, resolves one object per voxel. This is because the pixel color of each rendered polygon overwrites any color information that was stored in this specific pixel before. Therefore, the last polygon's color which is rendered into a pixel will be the one which defines the resulting object ID. If the depth test is enabled, this will be the object whose fragment in the slice is next to the viewer.

5.4.4 Identification of Multiple Objects per Voxel

The above used method for color coding can encode and identify 2^{24} object in the scene but resolves only one object in each voxel.

In a different approach to color-coding and rendering, the maximum number of objects identifiable in a scene is decreased to 24, but all 24 objects can be resolved in a single voxel simultaneously. This approach maps each object ID to one specific bit in the 24 bit RGB space as in Color-Coding Algorithm 5.2 and uses blending and the stencil buffer to compute the composite ID information for each voxel. In the resulting color bit pattern, each set bit denotes the occurrence of the specific object in this voxel.

Color-Coding Algorithm 5.2: Identification of 24 Objects per Voxel

```
RGB = SetBitIn24BitColor(ID); ID = {1,2,..,24}
//e.g ID=5 -> R:0 G:0 B: 0000 0001 0000

emissive = RGB
ambient  = diffuse = specular = (0, 0, 0)

//Example result: R:0 G:0 B:1000 1000 1000
//                indicates IDs 4, 8, 12
```

The composite bit pattern in the final pixel color can be received by using an OR operation when blending. This would result in the following bit patterns in a four bit example with IDs encoded in brackets: `1000 (4) OR 0001 (1) = 1001 (1,4); 0001 (1) OR 0001 (1) = 0001 (1)`. Unfortunately, OpenGL only implements arithmetic ADD operations. When adding two fragments with the same ID/color – a common situation because one object is likely to render more than one fragment into a specific pixel – the resulting bit pattern would indicate a wrong ID: `0001 (1) + 0001 (1) = 0010 (2)`. Therefore, the behavior of the OR operation has to be constructed of available operations. This can be done by utilizing the stencil buffer additionally to the blend function. The procedure is as follows:

The global draw algorithm is modified to enable blending. The according blend function is set to `glBlendFunc(GL_ONE, GL_ONE)` which arithmetically adds the incoming fragment to the fragment that is already in the frame buffer.

To prevent objects from writing their color bit pattern more than once into the frame buffer, the eight bit stencil buffer is utilized to store the ID of the previously written object. The stencil test function `glStencilFunc(GL_NOTEQUAL, ID, mask)` tests, if the incoming ID is equal to the ID in the stencil buffer. The third argument of this function sets a bit mask to specify which bits are included in the comparison. The stencil operation `glStencilOp(GL_KEEP, GL_REPLACE, GL_REPLACE)` changes the state of the rendering pipeline to write the ID into the stencil buffer, provided that the stencil test succeeds. Thus the `not equal` test will fail for following fragments with this ID. If the test fails, the fragment is discarded. This procedure requires that all geometry belonging to a specific object is rendered coherently. Otherwise, a different object may write its ID into the stencil buffer, by this overwriting the still required information about the formerly rendered object IDs. The scene-graph setup described in the following section provides for coherent object rendering.

Distributed Objects

In an eight bit stencil buffer only five bits are required to encode the 24 IDs. The unused three bits can be utilized for *distributed objects*, objects that can not be arranged to group all geometry together in the scene-graph to render coherently. Recalling the scene-graph in Section 5.4.5, this is the case for all objects that did not receive a unique ID, thus the ones directly under the root node. These objects can be anywhere in the scene-graph and all receive the same default ID. These distributed objects now directly test and write on one specific bit of the stencil buffer, set by the third parameter, *mask*. For example, to test and write uniquely on bit eight the stencil function is set to `glStencilFunc(GL_NOTEQUAL, 128, 128 = 1000 0000)`. All other objects with correct identification will use `glStencilFunc(GL_NOTEQUAL, ID, 127 = 0111 1111)` and test the ID against the lower seven bits of the stencil buffer.

Rendering Algorithm

The scene-graph setup described in the following section provides for the correct assignemt of parameters to the nodes. For the stencil test, the global draw algorithm has to be extended to enable the stencil test and to clear the stencil buffer for each rendered slice. The enhanced rendering algorithm is described in Algorithm 5.2.

Algorithm 5.2: Voxelization with Multiple Object Identification

```
...
Enable(Blending, Stencil Test)
SetBlendFunction()

 for i = 0 to z - 1
    zplane2 = zplane1 + step
    DefineOrthogonalViewingFrustum( zplane1, zplane2)
    ClearFramebuffer()
    ClearStencilBuffer()
    DisplayFilled(scaledScene)
        RenderObjectColor(RGB=SetBitIn24BitColor(ID))
        StencilBufferTest(ID, mask)
    WriteFramebufferToMem(memory + x * y * i)
    zplane1 = zplane2
SendVoxelsToAnalyzer(memory)
```

5.4.5 Scene-Graph Setup

The voxelization process utilizes the scene-graph in a similar way as described for the Triangle Voxelizer in Section 5.3.2. Here, the two types of custom nodes (cpDCS, cpRoot) handle the switching between the normal rendering mode and the voxelization mode. This allows to use the application scene-graph for both the normal visualization and the spatial analysis of the scene simultaneously.

The cpDCS nodes are inserted above the top node of each object that should be identifiable in the volume data space (see Figure 5.11). Unique IDs are assigned to these custom nodes. In the voxelization mode this ID is mapped to an RGB color, which is then applied to the object's geometry with Color-Coding Algorithm 5.1 or 5.2. At the top of the scene-graph the cpRoot node is inserted. It ensures that the rendering parameters are set for all geometry and that objects which are not assigned a cpDCS node are nevertheless rendered in a single color, the *defaultID* color. The following pseudocode in Algorithm 5.3 shows the task of the cpDCS and cpRoot nodes.

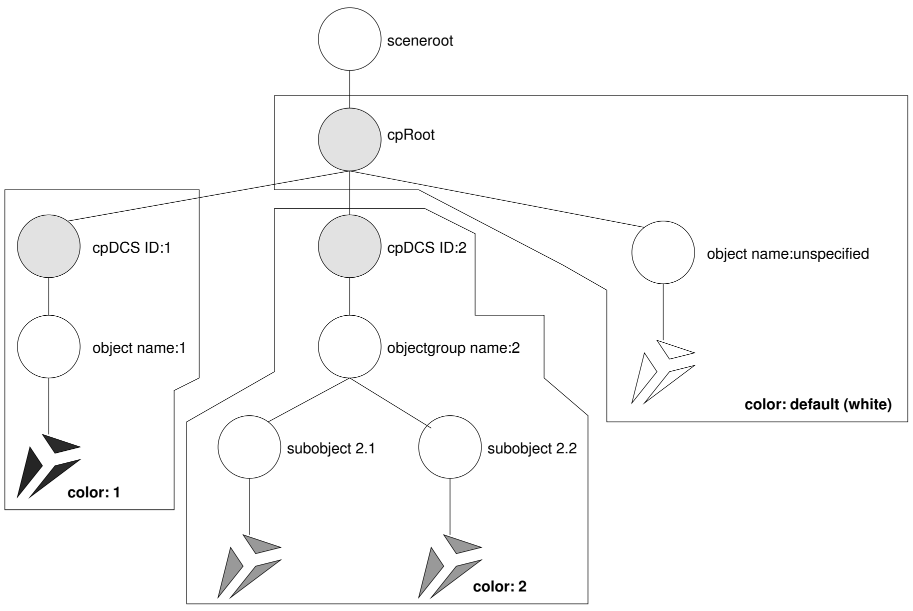


Figure 5.11: Scene-graph for Hardware-Based Voxelization. Nodes support the switching between voxelization mode and drawing mode. In this illustration the rendering flag is set to voxelization mode. The geometry is rendered in the color of the ID.

Algorithm 5.3: cpDCS and cpRoot

```
if cpRoot
   RGB = RenderObjectColor(DefaultID)
   if (MultipleObjectIdentification)
       RGB = SetBitIn24BitColor(DefaultID)
       glStencilFunc(GL_NOTEQUAL, 128, 128 = 1000 0000)
   else
       RGB=ConvertIDToRGB(DefaultID))
else if cpDCS
   if (MultipleObjectIdentification)
       RGB = SetBitIn24BitColor(ObjectID)
       glStencilFunc(GL_NOTEQUAL, ObjectID, 127 = 0111 1111)
   else
       RGB = ConvertIDToRGB(ObjectID)
```

To make Color-Coding Algorithm 5.2 work if multiple ”unspecified” objects are distributed over the scene-graph – thus polygons with the defaultID are not rendered coherently –, the parameter ID and mask are set different for cpDCS and cpRoot. This is further explained in the last item of the following Section 5.5.

If the matrix of a node changes, then this node automatically triggers the re-voxelization of the object. With the current algorithm this would result in re-voxelization and thus rendering of the complete scene. The voxelization time is dependent on the number of polygons that have to be rendered. Therefore, it makes sense to render only objects that need re-voxelization. This can be done by discarding the static objects from rendering with implementing a switch. This switch stops the traversal if the nodes geometry did not change. This process is in effect like *culling* static objects from the scene.

5.5 Implementation Issues

This section lists several issues that had to be dealt with while implementing the hardware-based voxelization for the SGI Onyx InfiniteReality 2 multipipe environment. Though some of the problems are very hardware specific, we nevertheless think them to be generally important for everyone considering the implementation of the described voxelization methods. The last two points listed are hardware independent.

Invisible voxelization rendering The hardware voxelization method needs a graphics context, a window, to render in. This can be a unique voxelization window or any available application window. The advantage of a uniquely dedicated voxelization window is that, if the system supports this, it can render in parallel to the application and it can request any framebuffer setting required for voxelization. The disadvantage is that it occupies some of the visible space of the screen. Thus, full screen applications require the voxelization process to happen in the available application window. This can be implemented by sequentially executing the application and the voxelization process. The voxelization process renders into the back buffer without swapping the buffer to the screen. Therefore, this is done invisible and independent of the framerate of the system. After all slices are rendered, the normal rendering – the rendering of the scene – takes place again. Only the result of this last rendering will be swapped into the front buffer, thus displayed. This results in the voxelization process being invisible although it uses the already available window. Another possibility is, if available, to use an off screen buffer for voxelization. Depending on its implementation, the off screen buffer may render in parallel to the normal rendering process, thus making the voxelization again independent of the application without opening a separate window.

Depth buffer Despite disabling antialiasing by pfAntialias(PF_OFF) in the voxelization process, the Onyx IR produces antialiasing-like artefacts, when two different colored polygons were close together, therefore having the same z value. This led to the assumption that these effects occurred when the Onyx dealt with z-fighting. The Onyx seems to implement some sort of blending for fragments with the same depth value, which produces antialiasing like artefacts. The solution was to disable the depth buffer for the voxelization draw.

The depth information can be discarded for voxelization because the exact position of polygons in each voxel is not required.

Multipipe considerations Section 5.4.5 describes the implementation of voxelization information into the application scene-graph. Depending on a flag defined in the scene-graph the scene-graph traverser sets the states in the rendering pipeline to generate images for the application visualization or for the voxelization process.

Multipipe environments fork multiple draw processes that simultaneously use the scene-graph. Thus it is possible that one draw process does voxelization while the other draw processes render the scene. Therefore, in multipipe or multi-process environments, it is not possible to use global flags in the scene-graph to switch rendering parameters for specific pipes or processes. Instead or additionally, the viewport or process information has to be considered, to decide on a scene-graph level, which rendering parameters need to be assigned.

Polygons small compared to pixels For very small resolutions of the voxel space – large voxels compared to the geometry –, complete polygons are likely to lie completely inside the screen space of a voxel, a pixel. They are then discarded by the graphics system and will not contribute to the final image. If polygons are explicitly rendered as lines or points they receive a defined minimum pixel thickness and are therefore always visible. The Algorithm 5.1 renders once in filled and once in wireframe mode to overcome the described problem. The result is that the scene-graph is traversed twice.

The extended voxelization method in Algorithm 5.2 requires that all geometry belonging to a specific object is rendered coherently. A second rendering pass would render the same object a second time, after the first pass is finished for all objects. If the stencil buffer for a specific bit was intermediately set to a different ID by a close object, then the object's ID is added a second time. This will result in a wrong ID in the final analysis (see Section 5.4.4). Thus, the color-coding method would fail.

A solution to this is to change the scene-graph for each object in a way that its sub-graph is rendered twice, once in filled and once in wireframe mode. This can be done by the cpDCS and cpRoot nodes.

5.6 Voxelization Time Experiments

To evaluate the hardware-based voxelization method, this section lists five experiments. They measure the voxelization time for varying resolutions of the volume data and for varying number of polygons. One experiment compares the hardware-based method to the software-based method. For all experimental data five values were collected and visualized.

The first experiment, Experiment 5.1 in Figure 5.12, evaluates the influence of the three axis of the volume data on the voxelization time. The analysis of the resulting volume is disabled to evaluate only the influence of time needed for creating the volume data. A difference is to be expected, because the x and y resolution – the image size – is handled in a different way than the z resolution – the number of slices to be rendered.

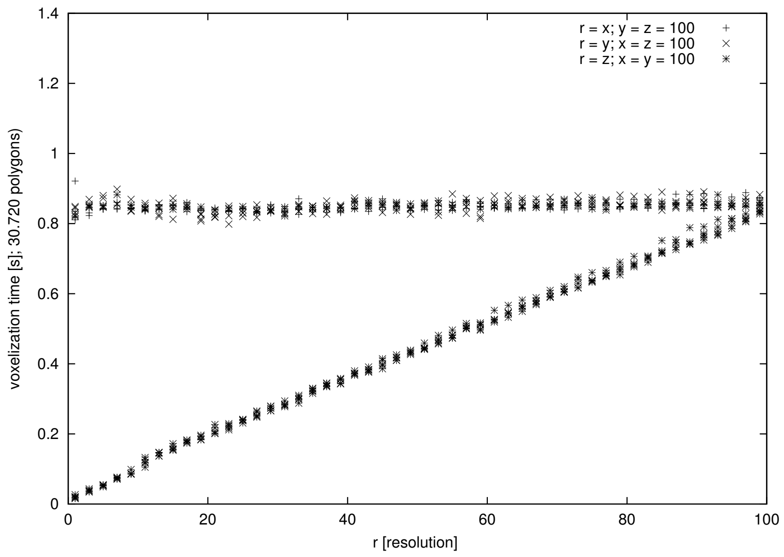


Figure 5.12: Experiment: Hardware-based Voxelization without Analysis of the Volume Data. Varied resolution, fixed no. of polygons (30,720). Measured is the voxelization time. Two axes of the voxel space are fixed, the third axis is varied.

Experiment 5.1:

Technique: Hardware-based voxelization without analysis of the resulting volume data.

Input: Scene with 30,720 triangles distributed over 10 objects.

Parameter: The resolution in x, y, and z direction. Two axes of the voxel space are fixed at a resolution of 100x100, the third axis is varied.

Platform: SGI Onyx MIPS R12000 Processor 8 CPUs InfiniteReality2E DIVO Video boards.

Measures value(s): Voxelization time in seconds.

Observation: see Figure 5.12. The voxelization time increases linearly with the resolution. The influence of the number of slices to be rendered - the z resolution - is on the SGI Onyx R12000 4 to 5 times larger than the influence of enlarging the image size by x or y.

The results of Experiment 5.1 show that the influence of the x and y resolution (their gradients are 0.0002) is negligible compared to the influence of the z resolution (its gradient is 0.0085). From this experiment it is learnt that, for efficient voxelization, the scene has to be transformed in a way that the smallest resolution of the voxel space maps to the z direction.

The next experiment, Experiment 5.2 shown in Figure 5.13, extends Experiment 5.1 to also include the analysis of the volume data, which is necessary to complete the voxelization process.

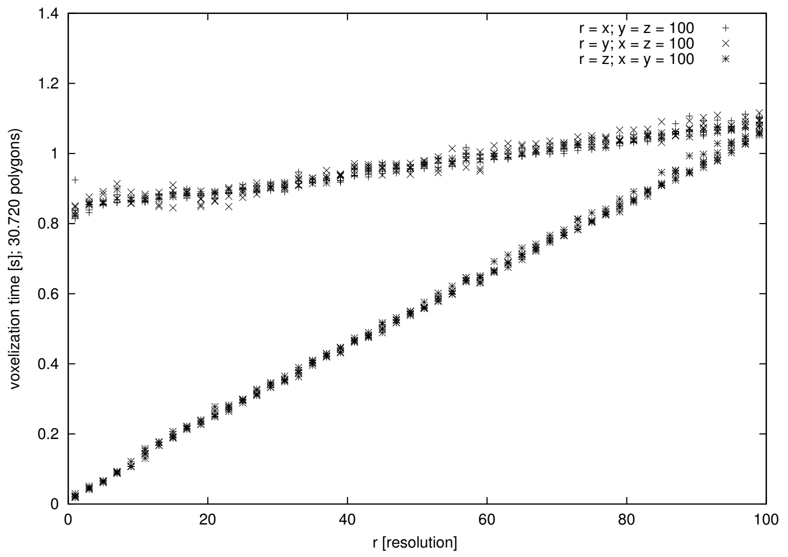


Figure 5.13: Experiment: Hardware-based Voxelization with Analysis of the Volume Data. Varied resolution, fixed no. of polygons (30,720). Measured is the voxelization time. Two axes of the voxel space are fixed, the third axis is varied.

Experiment 5.2:

Technique: Hardware-based voxelization with analysis of the resulting volume data.

Input: Scene with 30,720 triangles distributed over 10 objects.

Parameter: The resolution in x, y, and z direction. Two axes of the voxel space are fixed at a resolution of 100x100, the third axis is varied.

Platform: SGI Onyx MIPS R12000 Processor 8 CPUs InfiniteReality2E DIVO Video boards.

Measures value(s): Voxelization time in seconds.

Observation: see Figure 5.13. The voxelization time increases linearly with the resolution. The influence of the number of slices to be rendered - the z resolution - is on the SGI Onyx R12000 4 to 5 times larger than the influence of enlarging the image size by x or y.

The results of Experiment 5.2 show that the linear influence of increasing the volume data – each resulting voxel is analyzed – is equal for all three dimension. This offset is added to the figures of Experiment 5.1. A scene with 30,720 polygons can be voxelized in a resolution of $100 \times 100 \times 100$ in $1.05s$. For a resolution of $100 \times 100 \times 1$ the voxelization time is only $0.02s$.

To compare these results to the software-based method in Section 5.3, Experiment 5.3 varies the z resolution from 1 to 100 for both hardware- and software-based voxelization. The x and y *resolution* is fixed, because these axis do not have a significant influence on the voxelization time. Thus it is sufficient, to analyze the z resolution.

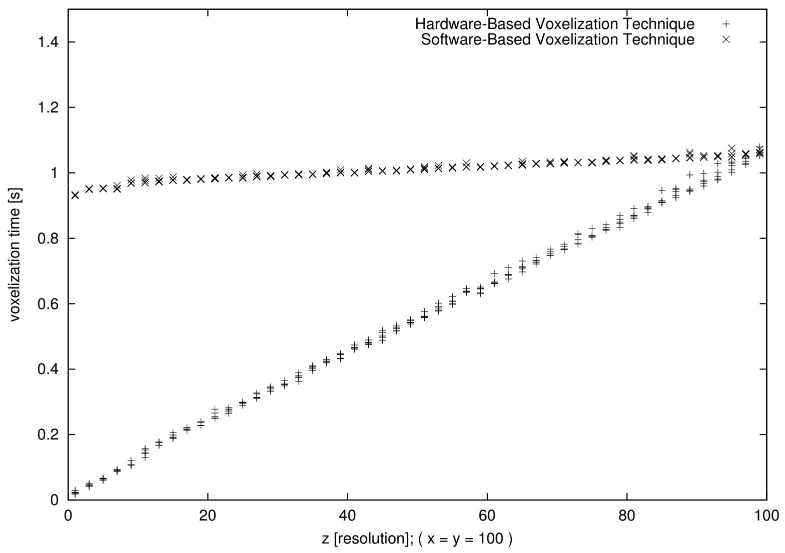


Figure 5.14: Experiment: Comparison of Hardware- and Software-Based Voxelization. Varied z resolution, fixed x and y resolution (100 x 100), fixed no. of polygons (30,720). Measured is the voxelization time.

Experiment 5.3:

Technique: Software-based voxelization and hardware-based voxelization with analysis of the resulting volume data.

Input: Scene with 30,720 triangles distributed over 10 objects.

Parameter: The resolution in z direction. The x and y resolution is fixed to 100x100.

Platform: SGI Onyx MIPS R12000 Processor 8 CPUs InfiniteReality2E DIVO Video boards.

Measures value(s): Voxelization time in seconds.

Observation: see Figure 5.14. The software-based curve has a large constant offset and a small gradient of the curve. The hardware-based curve has a minimal offset and a gradient of approximately one with increasing z. Both curves meet at $z \simeq 100$.

The results of Experiment 5.3 in Figure 5.14 show that, on the used hardware, the hardware-based method is faster than the software-based method, if the z resolution is smaller than 100. It is observed that the software-based curve shows a large constant offset but has a small gradient of the curve for increasing volume data resolutions. The hardware-based curve has a minimal offset but a gradient of approximately one with increasing z.

The experiments before were done using the color-coding algorithm which identifies one object per voxel. To evaluate the influence of the multiple object identification algorithm on the timing figures, both methods are compared in the next experiment.

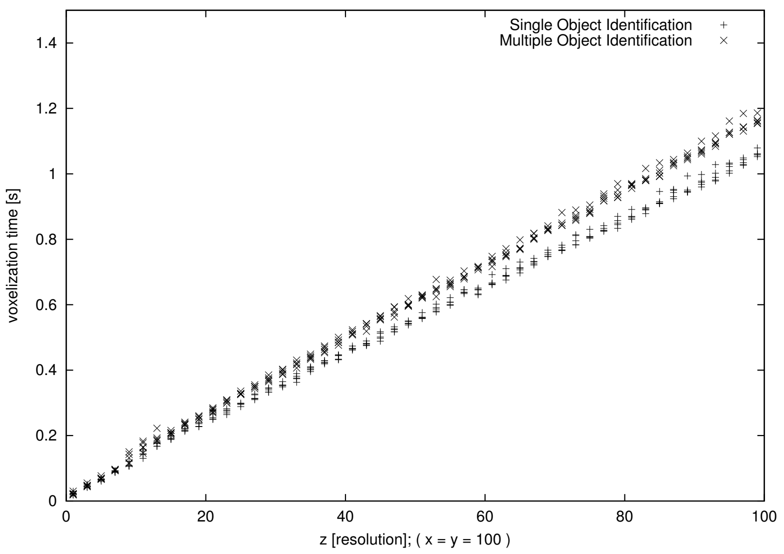


Figure 5.15: Experiment: Comparison of Single- and Multiple Objects Identification Algorithms for Hardware-Based Voxelization. Varied z resolution, fixed x and y resolution (100 x 100), fixed no. of polygons (30,720). Measured is the voxelization time.

Experiment 5.4:

Technique: Hardware-based voxelization using single and multiple object identification per voxel.

Input: Scene with 30,720 triangles distributed over 10 objects.

Parameter: The resolution in z direction. The x and y resolution is fixed to 100x100.

Platform: SGI Onyx MIPS R12000 Processor 8 CPUs InfiniteReality2E DIVO Video boards.

Measures value(s): Voxelization time in seconds.

Observation: see Figure 5.15. The curve representing the multiple object identification is only slightly steeper then the single object identification curve.

The results of Experiment 5.4 in Figure 5.15 show that the method to identify multiple objects per voxel is only slightly slower than the method which only identifies one object per voxel.

The following experiment evaluates the influence of the number of polygons on the voxelization time. It also compares the figures for the SGI Onyx R12000 with a PC configuration.

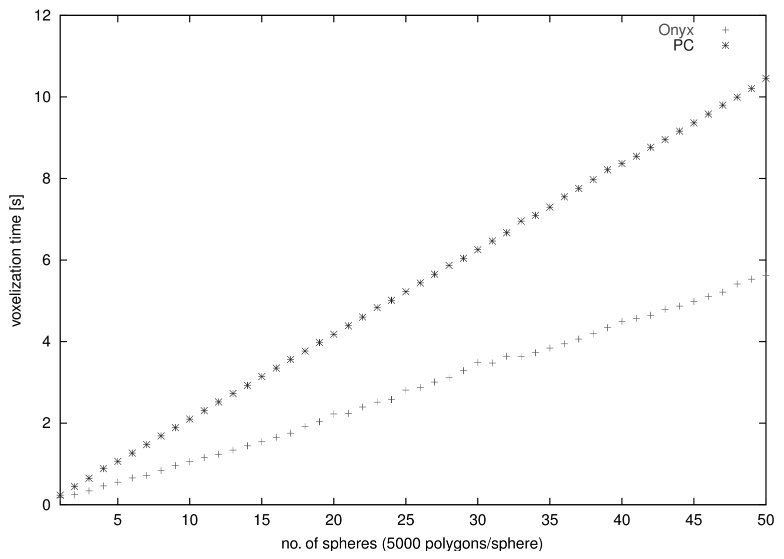


Figure 5.16: Experiment: Hardware-Based Voxelization for Onyx and PC. Varied no. of polygons, fixed resolution of $20 \times 20 \times 20$. Measured is the voxelization time. The amount of geometry is varied by increasing the number of spheres in the scene.

Experiment 5.5:

Technique: Hardware-based voxelization.

Input: Scene with n objects consisting of 5000 polygons each.

Parameter: The resolution x, y, and z is fixed to 20x20x20. The number of polygons to be rendered is varied by increasing the number of spheres in the scene. Each sphere consists of 5000 polygons.

Platform: SGI Onyx MIPS R12000 Processor 8 CPUs InfiniteReality2E DIVO Video boards. PC 800 MHz Pentium III PC with a GeForce 2 GTS graphics board and a 2xAGP bus

Measures value(s): Voxelization time in seconds.

Observation: see Figure 5.16. The voxelization time increases linearly with the number of polygons. The increase in voxelization time is steeper for the PC.

The results of Experiment 5.5 in Figure 5.16 show that the voxelization time increases linearly when increasing the number of polygons. Furthermore, it can be seen that the SGI Onyx is about 1.8 times faster than the PC configuration.

5.7 Discussion

Using the specialized graphics hardware for voxelization raises a number of issues. For example, antialiasing artifacts occurred because of z-fighting and polygons were not rendered, because they were smaller than the pixel size. Implementation techniques to solve the problems were presented.

The experiments showed that there is only a minimal decrease of the voxelization speed when the x or y resolution is increased. If the rendering of the scene is not fill-limited – not limited by how fast the graphics hardware can rasterize polygons for the required pixel resolution –, the voxelization time depends only on the number of slices to be rendered, thus the *z resolution*. Therefore, for efficient voxelization, the scene should be transformed such that the smallest resolution of the voxel space maps to the z direction.

Currently, the SGI Onyx IR still shows a better performance than the PC system in the experiments. But as the difference is so small it can be expected that with the next generation of graphics cards, possibly already with a 4xAGP bus, the performance of the Onyx can be realized with a low cost PC system.

The comparison between hardware and software-based techniques show that for large resolutions of z the hardware-based technique may become slower than the software-based technique. The choice of method depends on the type and requirements of the application: the degree of interactivity, a static or dynamic application, and the shape of the resulting volume data space. For static, non-interactive environments that do not require re-voxelization, the software-based technique provides an easy to implement voxelization method. The hardware-based technique is more complex to implement and needs a graphics window, but enables fast voxelization, which is required in interactive, dynamic environments.

5.8 Summary

Voxelization is a technique to generate volume data from geometric data. The volume data representation is a general data structure suitable for varies kinds of applications in the field of visualization, simplification and spatial analysis. We require it for the geometric setup of the potential field methods, introduced in the previous chapters. It is used for its uniform structure and for simplification of the geometric data.

This chapter presented two real-time capable voxelization methods for spatial analysis, a software-based and a hardware-based method. Real-time performance is important in dynamic environments, when frequent re-voxelization is necessary. Both methods generate object-coded volume data from geometric surface data utilizing the Performer scene-graph. Surface data is used, as the inside of objects is not visible, thus it makes no sense to include it into the calculation.

In the software-based method, each object's polygons are transformed into the voxel space which makes it an "object to voxel" approach: for each object the occupying voxels are identified. The hardware-based method is a "voxel to object" method, because the complete voxel space is rendered and then analyzed to retrieve the objects inside each voxel. A special approach to color-coding allows the identification of multiple objects per voxel, to make the results comparable to the results of the software-based polygon voxelization method.

For both methods performance measurements and a comparative analysis are presented. Performance data for varying resolution of the voxel space and for different complexities of the models are collected. They show that the hardware-based method is much faster for small resolutions of the voxel space in z direction, but may become slower than the software-based method for very large resolutions of z. The x and y *resolution* does not contribute to the voxelization time of the hardware-based method. The influence of the model complexity is linear in both cases.

Applications in the field of animation are often tasks, where the z *resolution* – the height of the space – is small compared to the x and y *resolution* – the large area that can be navigated. This utilizes the features of the hardware-based method in the best possible way. If only a small number of objects is dynamic, then the software-based method is preferred because of its simplicity and its real-time behavior for a small number of polygons.

6 Software Architecture

The aim of this work was to build a system which provides guided exploration for arbitrary applications which let users explore 3D scenes and incorporate simple object information. It is designed to be general and reactive to application and user input. The CubicalPath method can be used like a function call, with the attributes (camera data, geometry change, target change, interaction tool input) provided as parameters to the function. This calls for a software architecture which is independent of the specific application. A client/server architecture [Ede94] provides for this, as it separates the client and the server application. They communicate with each other through a well-defined and simple interface. This communication may also be done via a network, thus allowing the applications to be distributed.

Based on the methods described in the previous chapters, we developed a neutral software system which is designed to work as a server for any client application requesting guided exploration. The server itself is called *CPServer*. It communicates with the client application using CORBA [OMG, HV99], described below. A CORBA interface provides the necessary controlling functions for the client. This has the advantage that application and server can talk via a network and that different implementation languages and operating systems can be used. CORBA is also used for the communication between the CPAnalysisServer and the CPServer. The CPAnalysisServer is a server, which is used by the CPServer for analyzing the current view (see Section 4.2.1). The communication between the individual servers and the client application is illustrated in Figure 6.1.

6.1 CORBA

The Common Object Request Broker Architecture (CORBA), a three tier (layer) client/server architecture with an ORB (Object Request Broker) architecture that supports distributed objects, is an open standard for distributed programming with objects. It is a proposal by the Object Management Group (OMG), a group of over 500 members, including all major IT companies. It provides for object interaction between distributed object-oriented programs and different parts of the same program. It provides for unified communication between

- different hardware
- different operating systems
- different programming languages

The Object Management Architecture (OMA) consists of the Common Object Services (COS) and the Object Request Broker (ORB). The Common Object Services are standardized global services. The most important service is the naming service, which administers the objects and their location. This allows dynamic movement of objects. Each object registers itself with the naming service by providing its identification – a name – and its location. An application that wants to contact a specific object contacts the naming service via the Object Request Broker (ORB). It asks for a specific object by using the identifying name. It then receives a handle to the object location.

The interface to a CORBA object is defined in a language independent description the Interface Definition Language (IDL). It supports object-oriented concepts and is very similar to C++. Language elements are, for example,

- primitive data types (e.g. boolean, long, double, string)
- structs, unions, arays, lists
- sequences
- interfaces to objects
  - methods: return type, name, parameter list, exceptions
  - parameter handling: in, out, inout
  - global attributes: get, set
- and more which are described in [HV99]

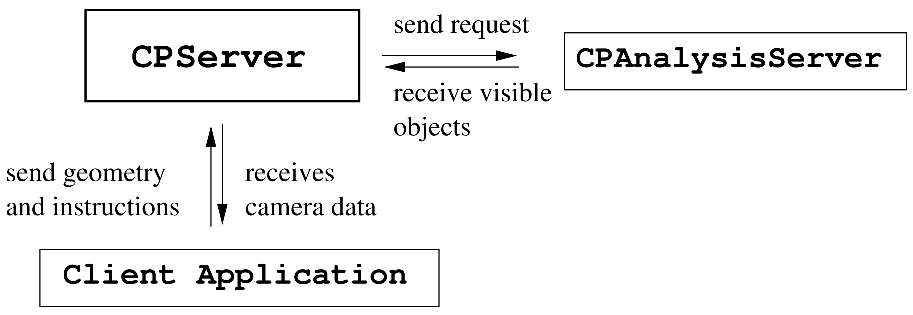


Figure 6.1: CubicalPath Program Structure. Communication of clients and servers via CORBA.

The interface to the server object is defined in the IDL. The CORBA implementation for the used language builds a skeleton class in the specific language. The server object then implements this skeleton class. At runtime, it registers itself with the CORBA system.

The client side object, the client application, contacts the server object via the name service. Or it can directly contact the object, once it has received and stored the server object reference, a string. With the server object reference it can call any function defined in the server's interface, as if it is part of the local program. This makes the communication completely transparent.

6.2 Interface to CubicalPath system

The client application controls the CPServer through a set of functions. It sends geometry data, attraction values for objects, sets repulsion and attraction functions, and modifies controlling parameters. Because default values are set for all functions and parameters, the application is only required to provide geometry data and attraction values, and control the start and stop of the camera data generation.

The rest of this section lists and describes the important control function of the CubicalPath system. The complete IDL file including the interfaces to the application and the CPAnalysisServer can be found in Appendix B.2. The below listed functions are the implementation of the interface to the CPServer which is called `iCPServer`. All functions are preceded by a `oneway`. This creates non-blocking functions call which means that the calling application does not need to wait for a reply.

initialization of a new CPServer session :
: To initialize a new CPServer session, the client application sends its CORBA object reference as a string to the CPServer:
`newSession(string objRef)`
This enables the server to directly send back results to the client handle without contacting the naming service and without knowledge of the client identification.

resolution of the cube space :
: The function `set_CPsize(long sizeX, long sizeY, long sizeZ )` sets the absolute resolution of the cube space, thus the number of cubes in each dimension. Together with the bounding box of the scene
`set_BBScene(CPVector min, CPVector max)`
this determines the physical size of one cube. The bounding box is automatically set by the CPServer, if not specified. Setting a specific bounding box can define the calculation space as any physical space of the scene. This can, for example, be a large predefined space, in which some objects move around.

transfer object geometry/cubes/matrix :
: The function `set_Geo_Tris(unsigned long ID, CPTris Tris)`

transfers the complete set of triangles for one object with its identifying ID. If the voxelization process takes place on the client side, then this function transfers the cubes coordinates encoded in the triangles. The function
`set_Transformation(unsigned long ID, CPMatrix m)`
sets a transformation matrix for a stored set of triangles belonging to a specific object. This reduces the amount of data to be transfered when a dynamic object is modified in the client application.

set attraction to objects :
The attraction value of each object is set with
`set_I(unsigned long ID, double attractionValue)` The CubicalPath system transfers this value to each of the objects's cubes.

set camera data :
The initial camera and any changes to the camera not introduced by the CubicalPath system needs to be communicated to the CPServer by the function
`set_Camera(CPCameraData cam)`

start/stop iteration :
The function
`runIteration(long n)`
starts n iterations of the CPServer, thus the calculation of n steps of the camera. The value (-1) initializes unlimited iterations. The function `stop();` stops calculating further iterations.

general control commands :
There are two control functions which set predefined functions or change specific values:
`send_command(string command);`
`send_command_withValue(string command, double Value)`
For example, the call `send_command_withValue(”attractionFactor”, 0.5)` sets the attraction factor γ_{attr} in Equation 3.25 to 0.5.

6.3 Client Application Setup

There are two ways for the client application to receive new camera data. The first way, a pull procedure, is that the client calls the CPServer system with a function `cam=get_camera()` to collect the current data. This means that the client asks for new data whenever it is required. The drawback is that the client has to wait, until the call returns with the results. This can take a while if the CubicalPath system has to generate new data or if the connection between the client and server is slow. If a continuous stream of camera data is required, this means continuously calling the CubicalPath system, thus blocking the execution of the client application.

The second way is to implement a push mechanism from the CubicalPath system to the client. The client initializes the CPServer with the function `runIteration(long n)`. The CPServer then calculates continuously, n times, new camera data and sends them back to the client immediately when ready. Meanwhile, the client – and by that the user – has full control over the application. This mechanism requires the client application to register itself as a receiving server for the camera data. Therefore, it has to implement the application interface, called `iCPApplication` in the IDL, thus implementing a server object itself. The server then calls the function `set_Camera(CPCameraData cam)` on the client. This is again a `oneway, void` function, thus a function that can be sent into the network without waiting for a result or the client to receive the data.

In the interface definition, the camera data is defined as a struct (see Figure 6.2) and is comprised of camera position, direction, and up-vector. It also has a center value – a look at position – and an ID. If any object is currently in the view, then the member ID holds the ID of this object. If there are more than one objects in view, then the object in the center is stored in the member ID. This value might be used by the client to initiate the display of further information about this object or for any other use in the application.

```
struct CPCameraData{
        CPVector pos;
        CPVector dir;
        CPVector up;
        CPVector center;
        long ID;
};
```

Figure 6.2: Struct Camera Data

6.4 System Design

The design decision for implementing the CubicalPath system as a server system and for using CORBA as a communication protocol was chosen based on several factors. An important point is that the CubicalPath system can run independently and on any machine. If included in the application program, it would significantly disturb the execution of the application unless implemented in a separate thread. The additional advantage in this context is that the server can run on a different machine, not disturbing the client application. It can even run on a machine that is connected over the internet. Another reason is that, with the IDL file, a simple software independent interface to the CubicalPath system system is defined and communicated. The client application can be implemented on any operating system in any language that supports CORBA. This makes the CubicalPath system an independent tool, which is controlled by a simple parameterization through the interface functions.

6.4.1 UML

The class design of the CubicalPath system was done using the Unified Modelling language (UML). The UML, together with supportive tools like OEW [OEW] or Rational Rose [Rat], help with the design of object-oriented programs. The tools provide a visual user interface to a program in the process of planning, implementing, and modifying, while the UML itself provides a convention for the different diagrams illustrating these stages. It is possible to design a class structure visually, and let the support tool automatically generate the source code for the class. Another feature of the tools is the ability to reverse engineer existing source code, to generate UML diagrams for the source code. With this, direct changes to the classes or their structure can be imported and visualized.

6.4.2 Class Design

Figure 6.3 shows the class structure for the CPServer. It consists of several classes, which are used by the CPServer class. Any geometric object with a unique identification provided by the application is stored in a CPGeoObj. All these objects are collected in the CPGeoList. The cube space is administered by the CPGC class. Each cube q in the cube space is an instance of class CPConfguration. The CPServer creates an instance of the CPGeoList (*geoList*) and of the CPGC (*gc*). It also stores the CPCamera object (*camera*) and a CPControl object (*control*). The CPControl is fully illustrated in Figure 6.4. It functions as the *control center* for the CubicalPath system behavior. All variables, states and instructions which can be set or modified by the client application are stored in this control class. This makes the interfacing between the CPServer, which uses theses information, and the iCPServer transparent. The class iCPServer implements the IDL interface. It calls appropriate functions in the CPServer or directly in the CPControl class, when requests arrive via CORBA.

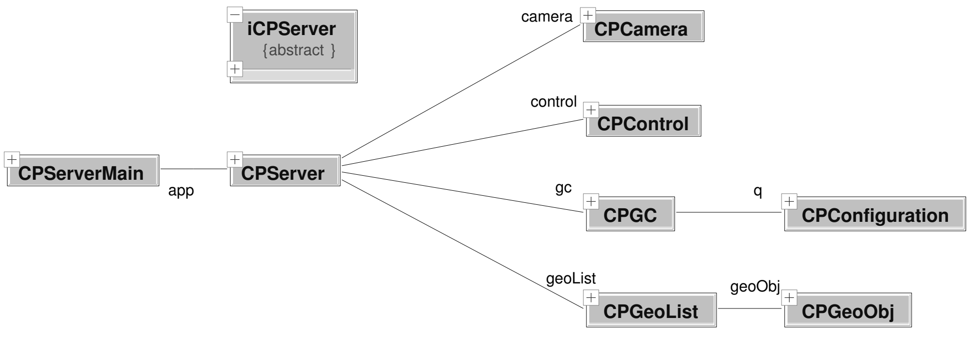
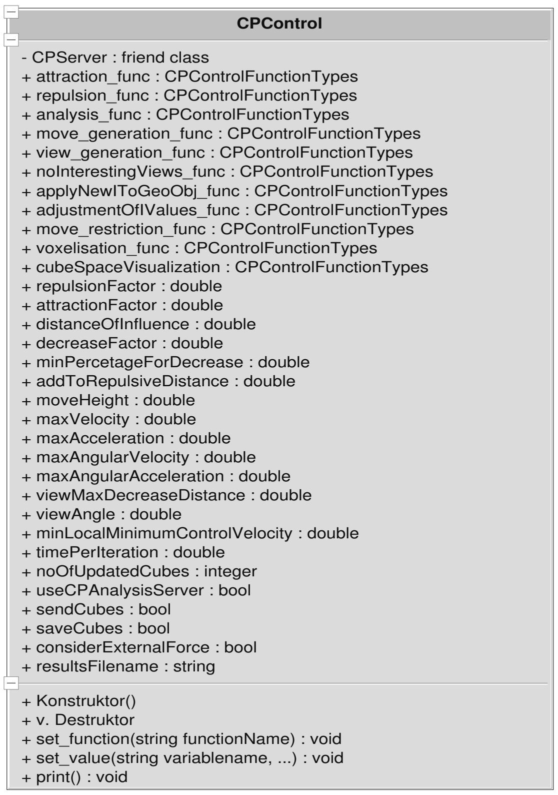


Figure 6.3: CubicalPath system Class Structure.



```
CPControl
- CPServer : friend class
+ attraction_func : CPControlFunctionTypes
+ repulsion_func : CPControlFunctionTypes
+ analysis_func : CPControlFunctionTypes
+ move_generation_func : CPControlFunctionTypes
+ view_generation_func : CPControlFunctionTypes
+ noInterestingViews_func : CPControlFunctionTypes
+ applyNewIToGeoObj_func : CPControlFunctionTypes
+ adjustmentOfIValues_func : CPControlFunctionTypes
+ move_restriction_func : CPControlFunctionTypes
+ voxelisation_func : CPControlFunctionTypes
+ cubeSpaceVisualization : CPControlFunctionTypes
+ repulsionFactor : double
+ attractionFactor : double
+ distanceOfInfluence : double
+ decreaseFactor : double
+ minPercetageForDecrease : double
+ addToRepulsiveDistance : double
+ moveHeight : double
+ maxVelocity : double
+ maxAcceleration : double
+ maxAngularVelocity : double
+ maxAngularAcceleration : double
+ viewMaxDecreaseDistance : double
+ viewAngle : double
+ minLocalMinimumControlVelocity : double
+ timePerIteration : double
+ noOfUpdatedCubes : integer
+ useCPAnalysisServer : bool
+ sendCubes : bool
+ saveCubes : bool
+ considerExternalForce : bool
+ resultsFilename : string
+ Konstruktor()
+ v. Destruktor
+ set_function(string functionName) : void
+ set_value(string variablename, ...) : void
+ print() : void
```

Figure 6.4: CPControl Class.

6.5 Programmer's view

This section lists the calls to the CubicalPath system which have to be made by any application in order to use the CubicalPath system. It supposes that the Cubical-Path system runs on a computer connected via a network or on the same computer and that the system is registered with the global CORBA naming service.

To connect a client application to the CubicalPath system the programmer has to

- implement the short `iCPApplication` interface, as listed in Appendix B.2.2.
- connect the client to the ORB.

For the initialization of the CubicalPath system the application is required to

- set the resolution of the CubicalPath system and the bounding box of the area, which should be observed by the CubicalPath system. This is the area in which geometry may reside during execution. This area can be larger than the scene space of the original scene file, if objects are dynamic like in the 3D Puzzle application in Section 7.1.
- collect and send the geometry, the polygons, belonging to each object that should be identifiable by the system.
- set the current camera data.

Then, by setting one object attractive and starting the calculation of camera data, the camera starts moving. No further input to the CubicalPath system is necessary. Section 6.5.2 explains possible ways to fine tune the behavior of the CubicalPath system by adjusting the attraction and repulsion functions to the needs of the application.

In dynamic application with dynamic objects or interactive user input, the client provides the CubicalPath system during runtime with

- the attractivity of specific objects, if it is changed by the client side due to changes in the interest of the user
- start and stop of the continuous calculation of new data
- any changes in geometry by sending new transformation matrices or a new set of polygons
- any changes to the application viewpoint/camera introduced by the user or the application itself
- any input vector generated by an interaction tool, if this feature is enabled for use with the CubicalPath system.

6.5.1 AVANGO client application

Applications at Fraunhofer IMK are typically written in AVANGO, IMK's virtual environment development framework described in E.4. In this system, geometry and functionality is stored in nodes of a scene-graph. We have implemented a node called *cpConnector*, which contains the above listed functions for connecting to the CubicalPath system, and can be inserted into the scene-graph. With this node, the CubicalPath system can be controlled in scheme, the scripting language used to write AVANGO applications. This makes interfacing to the CubicalPath system as simple as calling some predefined scheme functions. Modifications can be initiated on the command-line, if required. The commands are listed in Figure 6.5. The fields can directly be set. A change in a field is distributed to the CubicalPath system, if applicable.

The cpConnector node also receives the returning camera data and writes them into the field cameraMatrix. This field can be connected to the viewer matrix, such that the viewpoint of the viewer is driven by the CubicalPath system. The returning data is pushed into a message queue. This ensures that the application can collect this data whenever it returns from drawing a frame, and that it may take only the last one, if more than one value is in the queue.

One of the tasks of the client application is to transmit the objects' geometry as triangles to the CubicalPath system. AVANGO, which is based on Performer^{TM}, is able to read geometry data from a large number of file formats. This is done by the Performer File Loader. The most common formats are the Inventor (iv) and the

```
commands:
    set-BB(xmin, ymin, zmin, xmax, ymax, zmax)
    set-Size(cubesX, cubesY, cubesZ)
    set-Iteration(n)
    set-Attraction(objectsID, attractivity)
    set-Camera(posX, posY, posZ, dirX, dirY, dirZ)
    set-CameraCenter(posX, posY, posZ, centerX, centerY, centerZ)
    set-CommandWithValue
    set-Command
    send-Voxelization-Results

    do-software-Voxelization
    trigger-Scene-Update
    set-startTime

fields:
    cameraData, runIteration, clientSideVoxelizationFlag,
    softwareBasedVoxelizationFlag, TriangleUpdate
```

Figure 6.5: Scheme commands. Commands and fields to control the CubicalPath system with the cpConnector node

Flight (flt) formats. Performer internally builds a scene-graph from this file. The CubicalPath system requires a separate transmission of each of the objects with their corresponding identification. This is straight forward if each objects' geometry is in separate geometry files. If they are in one file and each object is assigned a unique name in the scene-graph – as described in Section 5.3.2 – then it is possible to identify each object by this name. This task is also automated by a special node, the *cpDCSReplacer* node. This node has the task of connecting each specific object in the scene-graph to the CubicalPath system, to initializing the system and to informing it of any changes. The connection and communication itself is done by the *cpDCS* node, described in detail in Section 5.3.2. It is assumed that the complete sub-graph, the graph under the named node, belongs to the geometry of this object.

A complete scheme call to the cpDCSReplacer node would look like

```
(-> cpReplacer 'replace-dcs object-list  cpConnection Scene-group)
```

where cpReplacer and cpConnection are the instances of the cpDCSReplacer and the cpConnector node, 'replace-dcs is the function name in the cpDCSReplacer node, Scene-group is the root node of the scene-graph that should be searched, and object-list is a list of object names which should be found and separated in the scene-graph. For example for the "room" setup in Figure 6.6, object-list could be ("table", "lamp", "chair", "UFO", "chestOfDrawers") if these strings are the names of each object in the scene-graph. The nodes and initializations described in this section comprise what is needed to connect the AVANGO application to the CubicalPath system and to trigger the initial and subsequent updates of the cpDCS nodes, thus the objects' geometry.

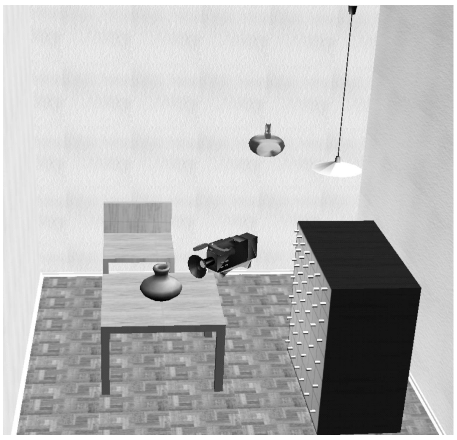


Figure 6.6: Example Scene: Room. This is the geometric version of the voxelized scene in Figure 4.2.

6.5.2 Fine Tuning

The behavior of the potential field algorithm largely depends on the attributes for attractivity α, the distance of influence for repulsion d_0, the scaling factors γ_{rep} and γ_{attr}, and more important the structure of a scene. A very crowded environment will behave differently to a wide open room. In a crowded environment it is important that the repulsion is adjusted such that the camera also can pass through a narrow passage. If the repulsion of objects or d_0 is too large, or the attraction value of targets behind narrow pathes are too small, then an visually open pathway may not be open to the camera. This requires adjustment of the potential field functions to each environment.

The following attributes can be changed for fine tuning:

- The distance of influence of obstacles d_0, the overall influence of attractive forces γ_{rep}, and repulsive forces γ_{attr}.
- The range of values α, $\alpha \in [0, \alpha_{max}]$, both for α_{pos} and α_{view}. The larger the value, the larger the resulting force.
- The maximum camera velocity v_{max}. This limits the maximum velocity, thus decouples speed from force.
- The maximum angular velocity $\omega_{angular}$. This limits the speed of rotation.

6.6 End-user's view

A user of a client application sees the CubicalPath system system through the interface provided by the application. This can, for example, be an interface to a knowledge base, controlled by text or speech input. The user would query the knowledge base which returns, depending on the results, a list of objects which are of interest in the current context. The application then sends theses results (object IDs and relevancy/attractivity, which form the goal field) to the CubicalPath system, which in turn generates camera motion that presents these objects. Thus, the users see themselves moving after they posed a query to the knowledge base. A condensed version of a query would be the direct selection of a single target from a list. This reduces the supportive exploration method to do automated travel. The camera then would move to the target object, or follow it, if it itself is moving.

6.7 Summary

The CubicalPath system is designed as an auxiliary supportive system. It uses a platform and machine independent client-server architecture based on CORBA. The virtual environment application and the application information space are connected to the server system through a lean interface, which mirrors their common data structures. In virtual environment applications, these data structures are the scene-graph for storing the geometry according to their objects and the transformation matrix connecting the viewer to the interaction tool (recall from Chapter 2 that a virtual environment makes not a lot of sense without being able to navigate/travel and this is normally done by connecting an interaction tool to the viewer of the application). For databases as the common representation of information spaces, the common output of a query is a list of results, based on the query. In our case, these are object IDs with an additional attractivity value, stating their relevancy to the query.

Through this interface, the CubicalPath system is provided with geometry, target and attractivity values, control parameters, and interaction tool vectors, if appropriate. Outputs of the system to the client application are continuously generated camera data which are connected to the transformation matrix of the client application viewer. This results in the CubicalPath system being easily integrated into existing virtual environment applications.

In this chapter, we have presented a programmer's view and an end user's view on the system. The programmer needs to connect the systems and their input and output parameters and may have to adjust control parameters to ensure a desired motion behavior. She has to modify the client application to be able to send and receive the required data. This was implemented for our virtual environment applications using AVANGO and described as an example in this chapter. The end user controls the exploration support solely through the interaction metaphors provided by the application and through querying the application information space.

7 Applications for Guided Exploration

Applications which require exploration of 3D data can be categorized in a) applications in which the structure and meaning of the 3D data is unknown, and b) applications, in which 3D data is arranged on purpose. In the first case, data is collected by a scanner, a simulation process, or other methods. Exploration helps to extract the relevant information hidden in the data. Examples of this type of application are MRI data (see Figure 5.1) and geoseismic data (see Figure 5.2), both described in Section 5.1.2.

We focus on the second type of applications, where the 3D data (for example, objects in a scene) is created, collected, or arranged to visualize information in 3D. The user then explores the resulting 3D data to learn something about the presented data. These applications can be extended by an expert in the field who collects and edits relevant information and connects this to the 3D visual representation. By this, these extended applications are especially designed to transfer edited information and to encourage learning.

The arrangement of objects for visualization can be sub-categorized into

- compound, dense objects like an engine (see Figure 7.1)
- a collection of disjunct objects in a specific context, like a museum or an architectural model.

Compound, dense objects, for teaching purposes, are often visualized in an exploded view as in Figure 7.1, to make normally hidden parts more easily visible. For this, the connections between these parts are released and the sub-objects are moved apart.[1]

[1]To transfer additional information into this view, 2D visualizations, like the one in Figure 7.1, can label the sub-parts with reference numbers or text. This is difficult in 3D stereo visualizations because 2D text does not fit well into 3D environments and is difficult to position it to ensure the correct allocation to the according object. Non-stereoscopic desktop applications can overcome this because they render 2D images of the 3D model, and thus can place text in an appropriate way [Pre98]. If information is not included within the visualization but is externally provided (speech, extra text windows, or indirectly through queries), then the according objects in the visual representation need to be presented. Ways to present specific parts in dense models

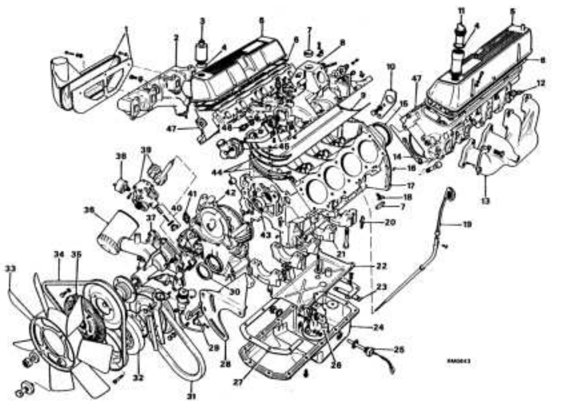


Figure 7.1: Explosion View. Rover V8 Engine.

A *collection of disjunct objects* for exploration is normally included in a context like a museum. Museums, for example, are explicitly designed to help the exploration into a subject.[2] Another context for a collection of disjunct objects is the use of large scale models of a town which can be used in travel guides to help tourist acquire a mental map of the town's spatial composition. In this context, the churches, museums, shops, etc. form a collection of possible attractions to a tourist.

This chapter presents three applications which make use of the CubicalPath system. Because the system is designed as a support system for arbitrary applications presenting 3D data, we took three existing applications and attached the CubicalPath system to them.

The first application, the 3D puzzle described in Section 7.1, is of the "compound, dense object" type of application which presents a model in an exploded and mixed up way. It is an interactive desktop application in which the parts of the model can be moved freely by the user. The second application, the virtual art museum in Section 7.2, presents artwork in the context of a museum, and is designed for an immersive virtual environment, the CAVE. The third application, described in Section 7.3, is the large-scale marketplace model of the city of Bonn, and is presented within the i-Cone. This application incorporates dynamic objects and lets the user influence the calculated way of the camera with an interaction tool. The latter two applications are of the "disjunct object" type. They restrict the movement

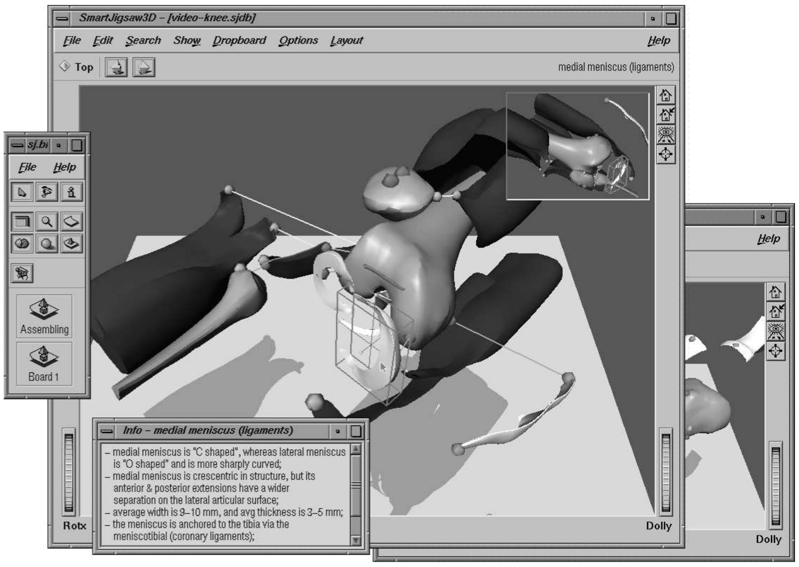
are to use a fisheye-technique [Fur86, Str98], to mark objects for presentation with a different color, or to make all other ones transparent. All these methods make changes to the scene itself, which may not be desirable in immersive virtual environments. This can be overcome by moving the sub-objects which are to be presented into the view without changing the scene itself. This is the approach we take to guided exploration.

[2] A museum brings together objects to give an overview of a certain field (e.g. physics, expressionist artwork, etc.) or to allow the objects to spatially and visually interact with each other. One example are architectural models, which consist of a collection of interior design objects, rooms, doors and windows. They are built to let users explore into their interaction and effect on each other. If these objects are presented out of their context, this interaction can not take place.

to movement parallel to the floor to prevent disorientation of the visitors in the immersive environments.

7.1 3D Puzzle

The virtual 3D puzzle (see Figure 7.2) has been developed at the Otto-von-Guericke University of Magdeburg by Ritter et al. to improve the understanding of spatial phenomena within a complex 3D model [RPDS00]. By enabling the users to assemble a geometric model themselves, the application motivates users to explore the spatial relationships. This gives users a goal to achieve while learning takes place. For this purpose, the 3D model of the subject at hand is enriched with docking positions which allow objects to be connected. Since complex 3D interactions are required to assemble 3D objects, sophisticated 3D visualization and interaction techniques are included. This application can display textual information for a selected item and is able to place a chosen object at the right position onto the partly-composed model.


SmartJigsaw3D – [video-knee.sjdb]
File Edit Search Show Dropboard Options Layout Help
Top
medial meniscus (ligaments)
sj.bi
File Help
Assembling
Board 1
Rotx
Dolly
Help
Dolly
Info – medial meniscus (ligaments)

– medial meniscus is "C shaped", whereas lateral meniscus is "O shaped" and is more sharply curved;
– medial meniscus is crescentric in structure, but its anterior & posterior extensions have a wider separation on the lateral articular surface;
– average width is 9–10 mm, and avg thickness is 3–5 mm;
– the meniscus is anchored to the tibia via the meniscotibial (coronary ligaments);

Figure 7.2: 3D Puzzle. 3D Puzzle application showing a scenario in anatomic education.

Guided exploration in the 3D Puzzle

While the main goal of the 3D puzzle is to provide direct interaction with the objects – the puzzle pieces – of the 3D model, there is also a need to provide help if the user gets lost. This may occur because of the large number of objects in the puzzle, the visual similarity of the objects, and because of user difficulties in navigating the 3D space. Navigation is especially important in order to explore the space, to have a close view on distant objects, to find objects which are occluded by others, and to find a suitable position for selection and transformation of objects. The navigation device used in this application is the Magellan Space Mouse^{TM}.

By extending the application with the CubicalPath system, these navigation tasks can be supported. The system is able to guide the user to a specific object or object group he or she is looking for, and it can show related objects.

Additionally, the CubicalPath system shows its strength by not only considering the spatial context but also by providing a means to incorporate the semantic context of the users' interaction. Each object in the puzzle maintains a value indicating its current importance (degree of interest – DOI) to the user. Based on the observation that most of the objects are related to each other in some way, the objects can pass or propagate some of their DOI to related entities. This semantic network is realized as a separate knowledge server tightly coupled with the 3D puzzle and the CubicalPath system (see Figure 7.3).

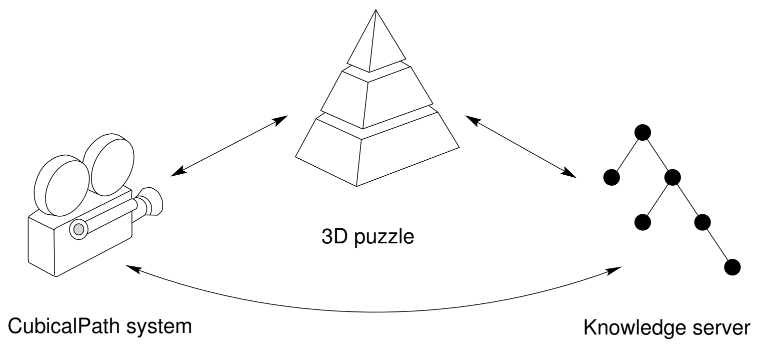


Figure 7.3: Cooperation 3D Puzzle and CubicalPath System.

User interaction in the puzzle causes a shift in this interest structure, which in turn is transferred to the CubicalPath system according to the goal to be accomplished. If, for example, the user requests help in the form of guidance to an object that has been selected from a list, this object gets a high DOI. Other objects in the current task context may also receive some of this interest. If the task is to place missing bones on a partly composed skeleton, this could mean assigning some DOI to other bones that are scattered in the immediate vicinity and which have to be docked on the skeleton nearby the originally selected bone. The CubicalPath system also presents these objects while guiding the camera – and thus, the user – to the requested object (see Figure 7.4).

The ability of the CubicalPath system to control the DOI itself leads to another useful feature. Since objects that have been shown to the user for some time lose some of their attractivity (the DOI decreases), the camera automatically pans to the next interesting objects if the user does not interact with the 3D puzzle. Hence, the system generates a dynamic animation reflecting the user's learning context.

Technical Details and Results

The 3D puzzle was modified to provide for the information required by the CubicalPath system and to be able to receive and use the resulting camera data. For the latter task, CORBA was incorporated into the 3D puzzle.

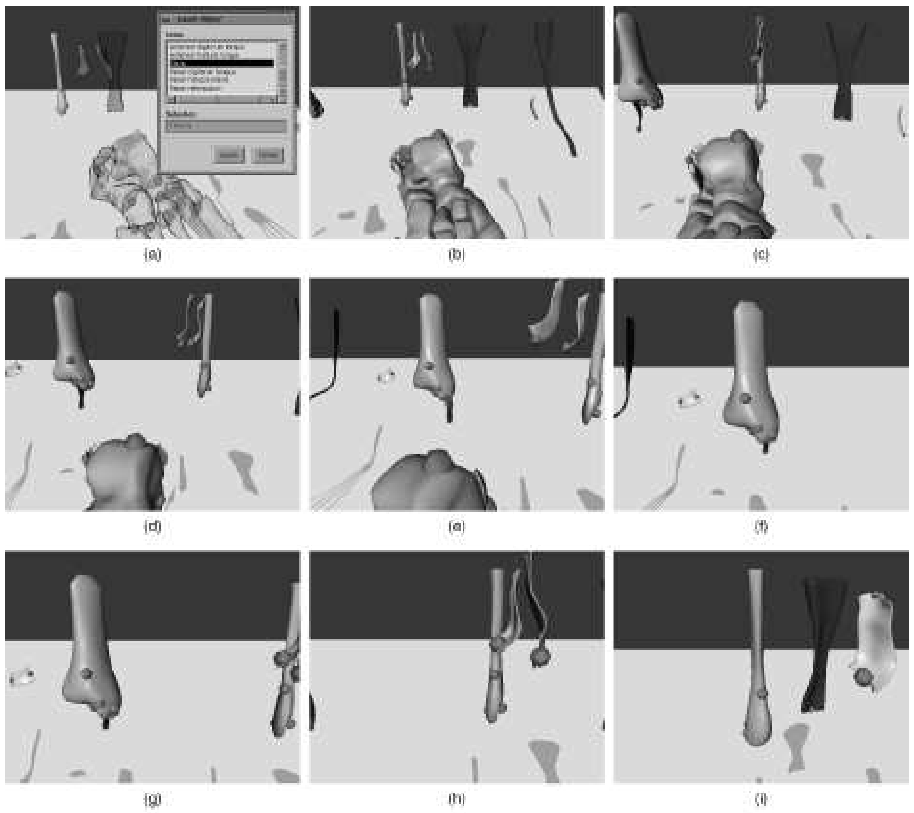


Figure 7.4: 3D Puzzle Animation. Presentation of two objects in the 3D Puzzle using the CubicalPath system. In (a) the user selects an object (fibula – the calfbone). The system then guides the user to this object in (i) and on the way also presents one contextually connected object (tibia – the shin)(f).

To setup the CubicalPath system, the 3D puzzle application transmits each object's geometry with an accompanying ID to the server. Additionally, it sets the resolution of the cube space and the bounding box of the space in which objects can be moved. After this, it transmits the results of a user query in form of a list of object IDs and their attractivity to the server. It also starts and stops the motion generation of the server. If objects are interactively moved by the user, the difference matrix to the original position of these objects is communicated to the server.

The movement in the 3D puzzle uses the complete 3D space. In a museum, the user normally moves through the space at fixed height, and therefore, the camera movement is typically reduced to 2D. Because we move in 3D space here, we occasionally had situations in which the camera approached an object from the top and remained in this position. This can be disorienting if normally the scene is viewed from a specific side. With the use of a navigation object which explicitly sets the desired position and direction for the camera, this situation was prevented. Now the camera moves until it stops at its specified final position at the side of the object.

In a first trial, we found the 3D puzzle and the CubicalPath system to work together well. The presentation of objects of interest succeeded well in the complex scene of the foot (recall Figure 7.2). The application also generated navigation objects (view/position pairs) for targets to enforce a special view on objects or sets of objects. This trial showed that, with the CubicalPath system, the objects could be presented in the desired way.

We tested the CPServer on an SGI Octane EMXI with 2xR12000 and 300MHz. The models of the foot consisted of 11,244 polygons for model A and 45,952 polygons for model B in the form of triangle strips.

The resolution of the cube space is defined by the number of cubes x, y, and z in these axes. The time v needed for the conversion of the triangles into the cube space – we used the software method described in Section 5.3 – depends on the size of the cube space. This is the time required to convert the complete model before the camera data can be calculated by the CubicalPath system. As shown in Table 7.1, the voxelization is time consuming in the presence of a large cube space. However, in the progress of guided exploration in dynamic models, it can be expected that relatively few objects will be modified at a time. This will dramatically reduce the conversion time when objects are modified or added, even in large resolutions of the cube space.

model	*resolution/cubes*	v/s	i/s
A	10^3	0.32	0.005
	25^3	0.39	0.03
	50^3	0.50	0.27
	100^3	0.88	2.13
B	10^3	1.18	0.005
	25^3	1.22	0.03
	50^3	1.42	0.27
	100^3	2.62	2.12

Table 7.1: Time measurements 3D puzzle. Speed of voxelization v and iteration i, time measurements in seconds. Model A consists of 11,244 polygons, model B consists of 45,952 polygons.

The calculation of one iteration i of the CPServer was finished within milliseconds in all but the last of these cases. This is sufficiently fast and results in a smooth camera path. In our example, the model was properly resolved by a resolution of 50^3 cubes.

7.2 Virtual Art Museum

The second application was originally created by Eckel at IMK as a virtual exhibition design environment in co-operation with the Kunstmuseum Bonn, the Museum of Contemporary Art of the city of Bonn [EB01]. The tool allows curators to interactively design art exhibitions in a virtual, real-size model of the museum space. Scanned pieces of artwork are mounted virtually on the museum walls, and the curators gain an authentic spatial impression of the exhibition in the CyberStage, IMK's four-sided CAVE. The CyberStage is a surround-screen, projection-based display system. Stereoscopic images are displayed from the rear onto the walls and directly onto the floor, and they can be viewed with shutter glasses. More about the CyberStage is explained in Appendix E.2. The exhibition design tool itself was implemented with AVANGO, IMK's framework for virtual environment development [Tra99].

Virtual exhibition design has proven to greatly simplify the exhibition production process. Curators can experience how pieces of artwork interact with each other long before they have been packed, insured, shipped, and physically mounted in the museum. Exhibitions can be previewed, and different designs and museum spaces can be compared easily, without the need of physically moving expensive and fragile artworks.

Guided exploration in the virtual art museum

The original virtual art museum application has proved to be a visually appealing and useful application to expert users. The objective was to make the final virtual exhibition accessible to non-expert visitors of the museum and to include additional help which would normally not be available in a real museum. For this, we extended the application with a knowledge base of information including attributes and features of the artworks. One part of the interface to this knowledge base displays all information of a certain selected artwork. Figure 7.5 shows a subject using this part of the interface in the virtual art museum. A second part of the interface allows the selection and cross-combination of attributes, for example the period in which the artworks were generated. Querying the knowledge base results in a list of selected pieces of artwork matching the criteria. When one or a collection of artwork is selected the user can switch to the automatic motion mode. In this mode, these pieces of artwork are displayed by moving the users to the matching pieces. At any time, the users can easily stop the automatic motion and resume free navigation. Restarting the automatic motion mode will resume presenting the formerly selected artwork, if some remained unvisited.

With a tracked, joystick-like device, the users can navigate through the museum. Selection of a specific artwork is done by pointing to the artwork with the stylus, a pen-like tracked wand with one button. This can also be done by selecting a name

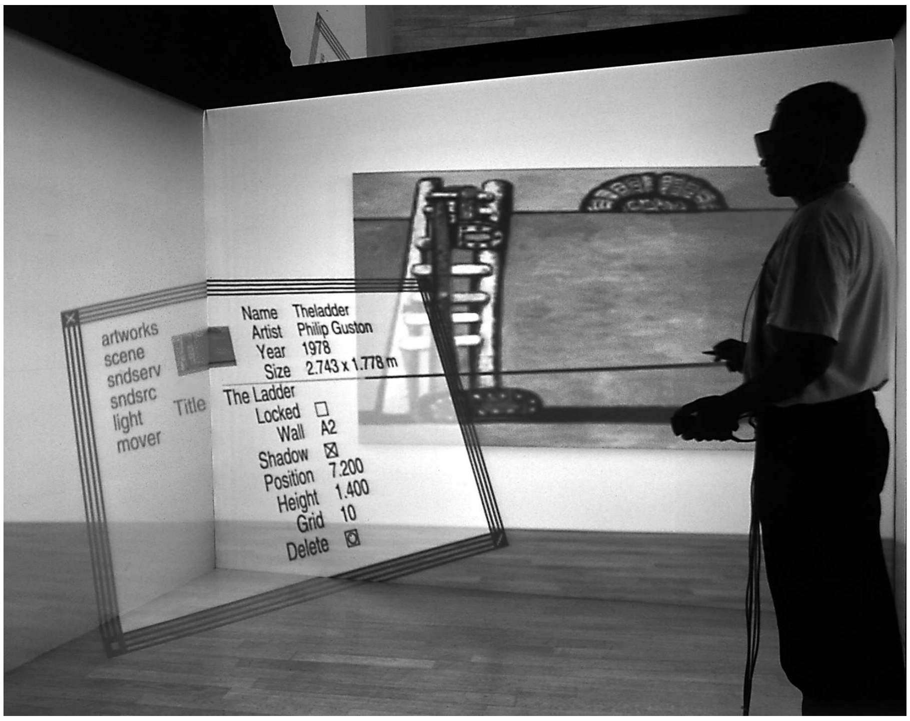


Figure 7.5: Virtual Art Museum.

from the list of all pieces of artwork. The users can request to be automatically moved to the selected artwork.

Technical Details and Results

In the initialization step, the geometry of both the museum and of the pieces of artwork is transmitted to the CubicalPath system and the desired resolution of the cube space is assigned. The bounding box of the space is automatically calculated from the space covered by the museum walls. After the initialization, information about the attractivity of a certain artwork and start/stop information is transmitted. If pieces are moved, their new transformation matrix is required. The current viewing matrix is transmitted in case a user or the application itself has modified the view. All the user requests are sent to the CPServer, the camera data generating server. In the automatic motion mode, the CPServer continuously sends new camera data to the virtual museum as long as there are interesting pieces of artwork to be visited. When all pieces are shown, the camera will stop.

For this application, AVANGO nodes which work as an interface to the CubicalPath system were designed and written. They are described in Chapter 6. Once these were available, it took a few hours to see the first movement of the camera in the museum, controlled by the CubicalPath system.

Because movement in the virtual art museum application is restricted to the floorspace, the z-component of the resolution of the cubes space can be set to 1.

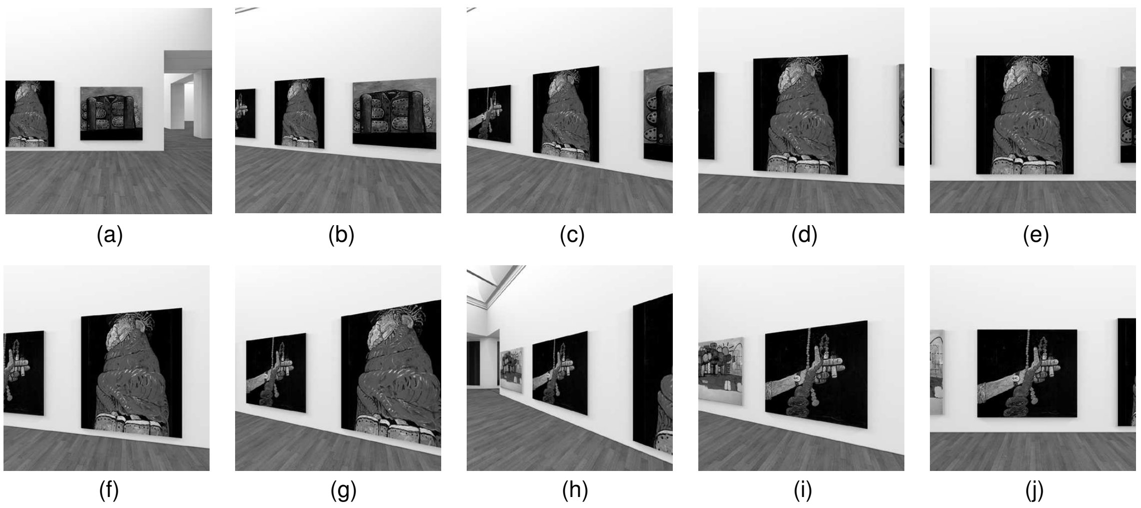


Figure 7.6: Virtual Art Museum Animation. Consecutive presentation of two pieces of artwork.

With the current application and a resolution of 40*40*1 cubes, we are able to produce new data at about 12 fps. The CPServer does not employ any interpolation of the camera data.

To ensure that a piece of artwork is viewed from a suitable position, this application employs navigation objects (see Section 4.4). The position of the target is set to be in front of the artwork at a distance of about one meter. The target for the direction is positioned in the middle of the artwork itself.

Figure 7.6 shows images of a generated presentation sequence[3], presenting the two pieces of artwork shown in (e) and (j). They are both attached to the same wall, situated left of the piece that is shown in (a). The animation remains shortly at the position (e) and then proceeds to (j).

7.3 Bonn Marktplatz

The third application shows the large-scale model of the Bonn Marktplatz in the i-Cone (see Figure 7.7). The Bonn Marktplatz is a large place in the town center of Bonn. The 3MB model of the marketplace covers an area of about 4,000 square meters. It has 974 objects, using only 32,000 polygons in total. Photographs were used as a source for the models texture maps. The i-Cone is a 230° surround display system with a diameter of 6.3 m. Up to 30 people can concurrently participate in a virtual environment presentation in this display. (The i-Cone is further explained

[3]Two videos present the movement through the museum. The first video presents the application in the CyberStage. The second shows the behavior of the CubicalPath system in the museum recorded directly from the output of the rendering pipe. The videos can be accessed at http://viswiz.imk.fraunhofer.de/∼steffi/videos

in Appendix E.3). This application can be explored by moving through the model at a fixed height using a flying joystick.

Guided Exploration on the Bonn Marktplatz

The large-scale geometric model does not incorporate any additional information about the buildings and shops in view. We collected and stored such information in a simple database. This information was of the type: name of object, feature (e.g. "Mc Donalds", "Food"). The CubicalPath system was attached to the marketplace to present this information by moving the user to the results of his or her query.

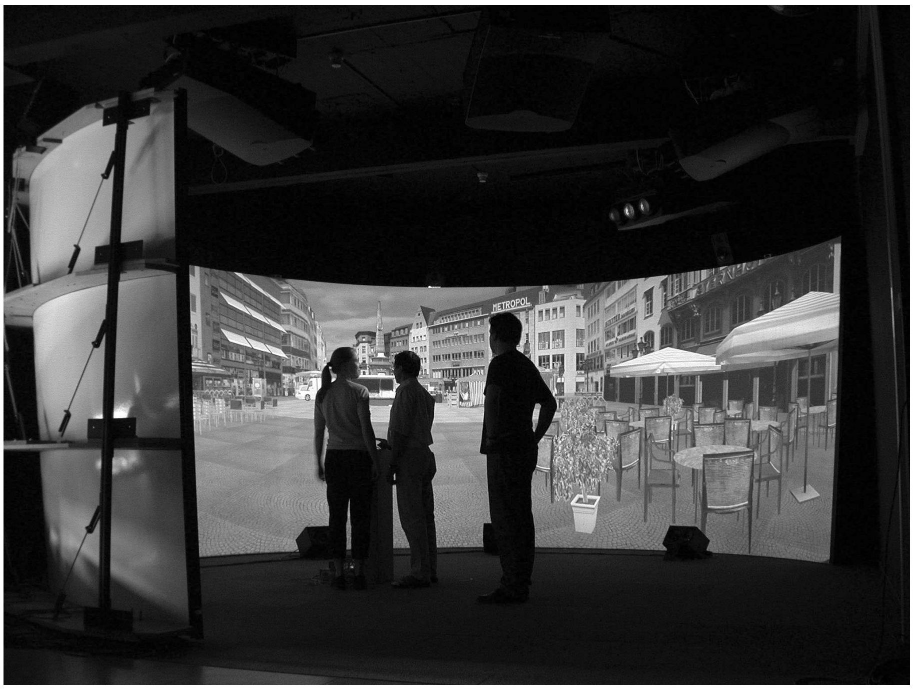


Figure 7.7: Bonn Marktplatz Application.

Technical Details and Results

In this application, the motion is parallel to the ground, as in the previous application. It differs from the virtual art museum application in that many objects (stands, lampposts, etc.) are in the way and moving objects (dynamic objects like cars) require the dynamic re-calculation of the cube space.

The resolution of the cube space was set to 283x189x1. This results in a square bounding box with a cube size of 33 cm in x and y and 150 cm in height for each cube. With this resolution, it is possible to resolve small objects like a lamppost, or to make movement through small pathways for a human-sized camera possible. For this application, the hardware-based voxelization method (see Section 5.4) was used to provide a suitable voxelization response for the dynamic objects. One resulting image of the voxelization process, which is used for analysis of the distribution of objects in the area, is shown in Figure 7.8.

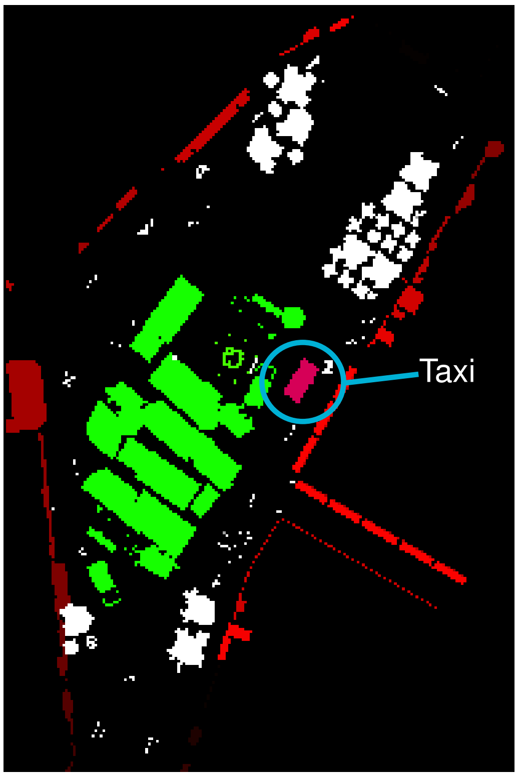


Figure 7.8: Bonn Marktplatz Hardware-Based Voxelization. This image shows the voxelization results of the state illustrated in Figure 7.9B. The shape of the moving taxi is indicated by the circle. The resolution of the voxelized data is 283x189x1. Objects which are selected for identification are rendered in different colors.

The CubicalPath system generated new camera data at 25 fps. For the dynamic case, this slowed down to 10 fps because of the voxelization and update time of the cube space. A video[4] presents the movement through Bonn Marktplatz. The video is divided into five parts. Part one introduces the i-Cone and its features. Part two shows two users moving manually through the model using the flying joystick. Part three shows the automatic motion generated by the CubicalPath system after the users selected "ice cream" as a query to the system. After this selection, they are moved to the nearest ice café. In this part, a second window appears on the video showing the same scene overlaid with a visualization of the current cube space. Red cubes indicate obstacles and yellow cubes indicate attractive targets. This is also illustrated in Figure 7.9A. The motion ends in front of the ice café, where the yellow cubes gradually loose attractivity (visually they become more and more transparent) until they become simple obstacles. Here the movement stops. In part four, the "Rathaus" is selected as a target. While moving to the target, a dynamic object, a taxi, moves into the way and the direct motion is disturbed to prevent colliding with the car. The cubes forming the moving car are a little behind the

[4]the video is accessible at http://viswiz.imk.fraunhofer.de/~steffi/videos.

actual shape of the car (see Figure 7.9B), because the visualization of the cube space is delayed by a few frames. The calculation of the cube space data itself is accurate. Part five shows the following of a dynamic target. Again, the users selected "ice cream" as a query, and the view starts following a moving ice cream truck until it comes to a standstill. Then the camera moves towards the window side of the car and views into it. This happens because the ice cream truck target was set with a navigation object. The final position is at the side of the car and the view direction target is inside the car.

With the CubicalPath system, it is possible to overlay the output of the system with user input. With a flying joystick or any other interaction tool, the user can manipulate the viewpoint. This can be overlayed with the output of the CubicalPath system as described in Chapter 4.3.1. We used this in the Bonn Marktplatz application for moving out of local minima and for influencing the motion generated by the CubicalPath system in a user defined way. Small forces generated by the interaction tool resulted in a combined output of the CubicalPath system and the user defined direction of motion. Large forces overrode the output of the CubicalPath system which resulted in direct manipulation of the view by the user. The manipulation by the interaction tool faded smoothly in and out the generated motion which led to an intuitive use of co-existing interaction tool and CubicalPath system.

7.4 Discussion and Summary

This chapter introduced three applications which made use of the CubicalPath system. These applications required several features of the system, listed in Table 7.2. In these applications, all features of the CubicalPath system presented in former chapters were utilized and tested. The first application uses an Inventor scene-graph while the other two use AVANGO, which is based on the Performer scene-graph.

	3D Puzzle	Virtual Art Museum	Bonn Marktplatz
2D motion		X	X
3D motion	X		
multiple targets	X	X	X
dynamic objects	X		X
dynamic targets	X		X
dynamic camera	X	X	X
interaction tool			X
sparse 3D space		X	
dense 3D space	X		X

Table 7.2: Application Features.

Motion in 2D, dense environments introduces more difficulties for the algorithm then does motion in 3D. If the camera has to pass an obstacle on the way, it is more likely that the camera finds a way around if it has three dimensions in which to choose a way[5]. Unless the obstacle in the way is equally dimensioned in all directions, the probability that the forces will pull the camera around the obstacles is higher in 3D. This problem occurred in the Bonn Marktplatz application. In some cases, for example, with a large group of chairs forming a convex solid shape, the camera moved into a local minimum between the chairs. Enabling the 3D motion resulted in the camera flying over the chairs, instead of stopping. We used manual movement to get out of these cases, because the users already had a navigation device, the joystick, in their hand. With the co-existing use of an interaction tool and the CubicalPath system, users were influencing the path in a way that moving into dead ends was avoided.

In desktop applications, there is a clear sense of front and center of display. In immersive display systems, like the CAVE and i-Cone, there is no clearly defined front direction, thus no clear direction of view for the user. In these displays, the user can look in any direction. A sense of front is important for the CubicalPath system, however, as the view motion direction is calculated for the midpoint of the screen. It is in this area that the current target is presented. We found that normally the users in the four-sided CAVE take the front screen (the one opposite to the entrance) as the direction of view. In the i-Cone, the mid 100° of the display is taken as front. People tend to focus on the vanishing point of fast motion – the direction of motion – in real and virtual worlds. If the motion is in a different direction than the closest, interesting object (i.e. if the direction of view is not the direction of motion), users tend to view in the direction of motion and not at the object intended to be presented. In the i-Cone this happens because they do not have a clear sense of front. This holds only for fast motion to distant objects. If the motion is slow, people focus more on what is to be seen, than where they are being moved. Also, if the object to be presented is close, thus covering a large part of the screen, this attracts the users' complete attention.

In immersive environments, the user is moved inside the object space. Being inside an exploded view, for example, may cause a loss of general overview of the complete object space. If this is important, a function "overview" may be used which moves the user out of the object to a "bird-eyes-view" of the complete object space. This could be done by specifying the camera position in some distance to the objects and pointing the camera in the direction of the center of all objects.

The CubicalPath system proves to be good for detail presentation of the objects, because it moves close to objects which are to be presented. In the museum, for example, the pieces of artwork are closely viewed. This makes users focus on details in this piece. Additionally, because they are presented in a large immersive environment, the neighboring pieces are visible to provide for the context.

[5]Technically, the algorithm does not choose a direction, but is pulled around the object by the forces behind the obstacle.

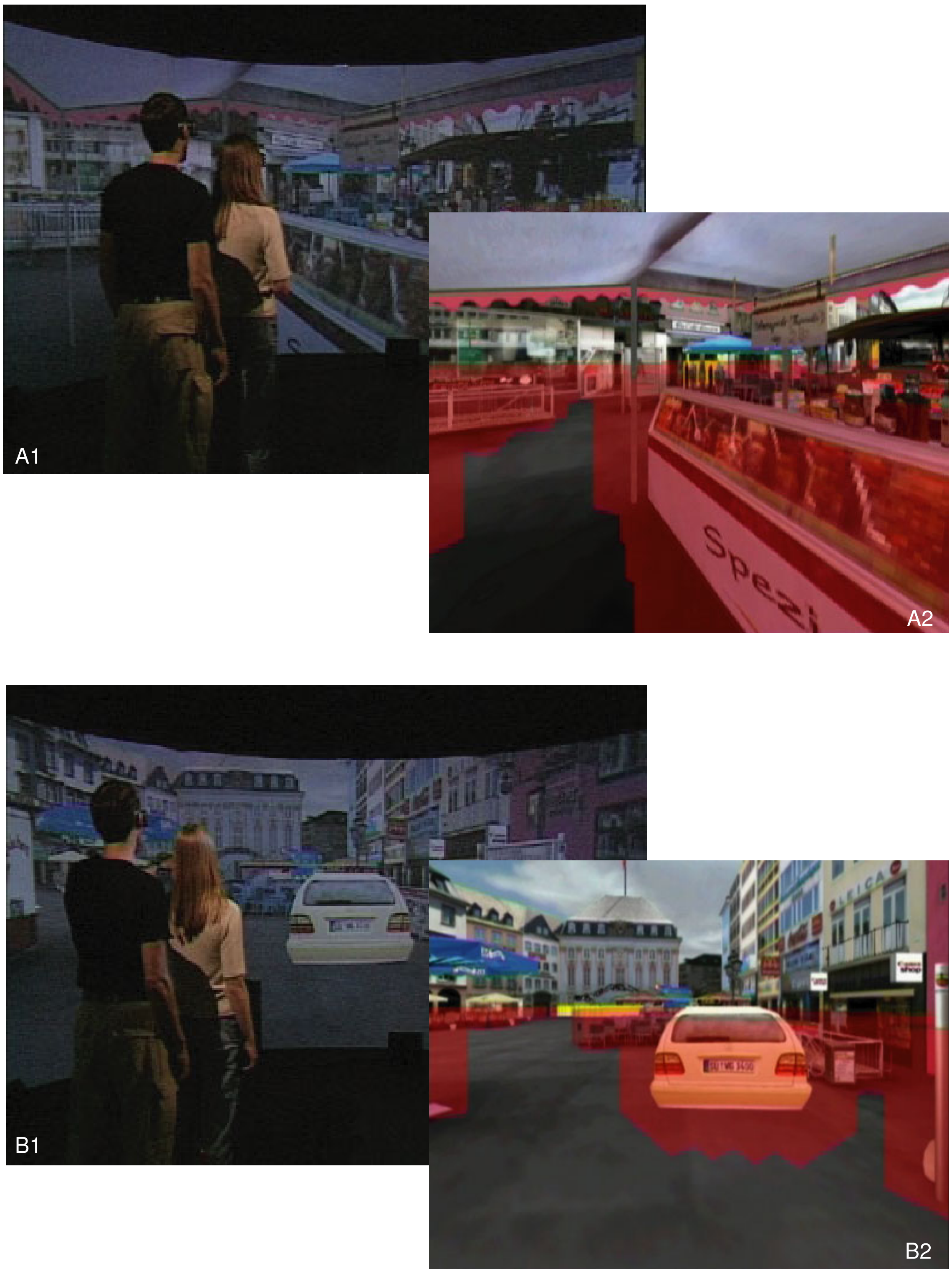


Figure 7.9: Bonn Marktplatz Application Overlaid with its Cube Space Representation. The cube space is visualized with red cubes indicating obstacles and yellow cubes indicating targets. A1 and B1 show the results in the i-Cone, A2 and B2 show the scene merged with the cube space. In A1,2, the users asked for "ice cream". The target, an ice café, is visible in the upper mid of the image. B1,2 show an dynamic object – the taxi – which moved in the way of the direct path to the distant building – the Rathaus.

8 Usability Study

The previous chapter described several applications that made use of the CubicalPath system. These examples showed that the system's approach to a camera data generation method is capable of moving the camera in the desired way. The method presents one or multiple targets one-by-one, deals with dynamic obstacles, targets and user input, generates the camera results in real-time, and provides an easy interface for the application programmer.

Though these application examples showed the usability of the CubicalPath system to us, it was not verified that this approach to guided exploration can enhance an untrained user's performance in exploring a scene and that its behavior is perceived well by users in an immersive virtual environment. To inquire into this, an informal user study was done in the context of the virtual art museum application, comparing the user's behavior, observations, and impressions with and without utilizing the CubicalPath system.

Measuring usability for evaluating the design of tools and devices for computer is a major part of Human Computer Interaction (HCI) research. The tools for formal tests were collected by Bortz and Döring in [BD95]. The authors explain how to perform classical, empirical studies and present all statistical methods for evaluation. Nielson [Nie93] focuses on all part of the usability engineering lifecycle and gives usable, everyday methods for testing and evaluation.

Usability testing is not commonly used in virtual environment research. This may have to do with the much smaller commercial community for virtual reality devices compared to that available for desktop or handheld computers, for example. Recent approaches to measure interaction techniques for travel [BDHB98] or for object selection and manipulation [BH99] were conducted by Bowman et al. In [BH99], Bowman and Hodges describe their eight parts of the design, evaluation and application methodology for evaluating interaction tools. They do an initial evaluation based on their experience and on informal user studies where they observe users or ask them what they think about a technique. Based on these observations, they build a taxonomy which splits each task into sub-tasks and their techniques. They then use their testbed evaluation, which joins the results of the taxonomy, outside factors, and performance metrices. A testbed defines a specific set of evaluations that theoretically test each important aspect of a technique for a given interaction task. The test results are then statistically evaluated.

In [HH93], Hix et al. evaluate navigation in a battlefield visualization virtual environment on a Responsive Workbench [KF94]. They list expert heuristic, summative, and formative evaluation as evaluation methods. Expert heuristic evaluation is conducted by an expert in the field who did not program the tested tool. Summative evaluation is an empirical comparison of different interaction tools by users. Formative evaluation collects qualitative data (e.g. narrative, in form of critical incidents while performing scenarios) and quantitative data (e.g. time needed). Hix et al. use these evaluations iteratively during the design cycle.

Our approach is an informal user study similar to Bowman's informal evaluation and Hixs's formative evaluation. Our method extends Bowman's informal evaluation in that it also considers outside factors (like user and system characteristics) and formally collects results through post-test questionaires [Rub94]. In addition to this, the tests are observed by test monitors (observers). Our goal is not to compare several interaction approaches to a certain task, but to gain insight into the usability and the deficiencies of the CubicalPath system in the context of a CAVE application.

The application we use, the virtual art museum application which is described in detail in Section 7.2, shows multiple pieces of artwork in the context of a museum. A database holds information about the artwork. This information can be visualized for a specific piece by selecting it. Or it can be queried to trigger the presentation of all pieces that match the query. The camera in this application moves in 2D as it follows the ground. This is the natural way of movement to people. And it is an easy to control flying model for untrained users. We chose this application for the user study as it can be supposed that every user is familiar with the exploration of real museums. In addition, the museum application covers a space which can be explored in a reasonable time but still holds several different layers of information (names, colors, attributes, groupings, etc.) which are not instantly visible. Also, users are not burdened by complex navigation tasks when they travel through the environment.

8.1 Goal

The user study is intended to gain insight into several aspects of the advantages and the usability of the CubicalPath system connected to the virtual art museum in the CyberStage.

One of the goals of the user study is to find out if guided exploration helps with understanding an exhibition better in a restricted time, while the CubicalPath system is seen as a supportive system which does not make the users feel deprived of their freedom. This is important as the process of free exploration and of self-learning should be supported. If users have the feeling of loosing control they may less feel being part of the virtual world and may also give up active involvement.

Another goal is to identify possible usability problems with the system and the interface. Problems here are suspected to largely influence the performance of a user in the exploration task.

Goal three is to find out whether the users' sense of presence in the environment decreases when they are automatically moved through the environment. The CAVE is an immersive display system, and therefore no parts of the application should detach the user from the immersive experience. Immersive in this context means the feeling of presence in the museum application. This feeling can be disturbed if unexpected or unpleasant behavior of the system is experienced, or if the user is left disoriented. Disorientation would be very undesirable because the spatial exploration and the creation of a cognitive map of the environment is the most important task in guided exploration.

The fourth goal is to inquire into the suitability of the method for producing pleasant presentations. Especially in immersive virtual environments, motion sickness can be a problem if motion is not initiated by the users themselves. This is an important question, as the approach does not allow smoothing of the results because of its stepwise, instant data generation.

8.2 Hypothesis

For the goals defined above, there are five hypothesises which should be verified by the test:

Hypothesis H1: Guided exploration improves the understanding of an art exhibition in the CAVE when time is restricted compared to when the user explores the exhibition himself. Guided exploration makes exploration faster and more effective.

Hypothesis H2: The CubicalPath system is experienced as a useful and supportive system which does not make the users feel deprived of their freedom to explore the environment themselves.

Hypothesis H3: The users do not feel disturbed or disoriented when they are automatically moved through the environment. Their sense of presence is not decreased.

Hypothesis H4: The CubicalPath system generates pleasant camera motion which does not cause motion sickness and which presents the targets in a useful and desirable way (as defined by the user).

In addition to checking these hypothesises which evaluate the CubicalPath system, also the usability of the user interface to the museum application needs to be verified:

Hypothesis H5: The application's user interface is designed in such a way that it is easily usable, supports the task, and reflects the user's expectations.

8.3 Test Design

The method we chose was an informal user study with 10 people. This method gives sufficient results to be able to verify the prior listed hypothesises without needing the time and financial resources of a formal user study.

8.3.1 Task Design

The task was designed to compare the user's performance in the museum application both with and without the support of the CubicalPath system. For this, two exhibitions, each with 16 different pieces of artwork in one room of the museum, were installed to provide for a comparable task setting with different content. A similar application setting is required for the comparison of the user performance (here the collection of content information) in both cases. Users should have to focus only on the content of the application rather than the application itself.

In the first stage, users had to explore one of the exhibitions for five minutes completely by themselves. They could navigate through the environment with a joystick and select an artwork with the stylus. With a button on the joystick, they could open a menu containing information such as name, date, and technique of the selected artwork. After about five minutes, the users were asked to write down any features, names, colors or other information they collected during this first stage. They were also asked to complete a number of general questions on a questionnaire.

In the second stage, users entered a different exhibition setting. Now, they were introduced to the supportive system. For a better overview of the possible combination of attributes that could be selected, a sheet with all this information was handed to the users. This sheet also described additional functions of the joystick and stylus. Again, the users had five minutes to explore the exhibition before they were asked what information they collected. They were also asked a number of questions concerning their experience with the system.

At both stages, users were introduced to the task and the tools and had two minutes to get used to them.

The order of the tasks – use the system first without the CubicalPath system then with it – was chosen for two reasons. First, the CubicalPath system supports the user in the learning task by structuring information with the help of the query interface and the presentation of the results. Once this structure is established, once the users are aware of specific features in the artists work which were selected by the curator or a teacher, this will influence their perception of the artwork and thus their performance in the self-exploration task. By letting them first self-explore the museum, users will come up with their own structure in this stage, which can then be kept or extended in the second stage. The second reason for this order of tasks is the overhead required to learn to control the system. The first task, the self-exploration task, requires the user to learn about the general menu control and the navigation control. The second task relies on the ability of the user to either self-explore or to use the CubicalPath system. The user needs to learn about the interface to the CubicalPath system in addition to the general menu control and the navigation tool. The proposed order distributes the learning overhead equally to both tasks – in the second part the general menu control and the interaction tools are already known – and therefore also distributes possible interface problems to both parts.

8.3.2 Questionnaire

The design of questionnaires for the evaluation of presence in virtual environments is discussed by Witmer and Singer [WS94] and Slater[Sla99]. Questions in questionnaires can have several forms [Rub94]: likert scales (LS: select agreement or disagreement with a statement), semantic differentials (SD: usually a seven point scale between a pair of adjectives), fill-in questions (T: free text answers), checkbox question (CB: select one of the preselected answers), and branching questions (B: if question A was answered YES, please answer question A1). We used all elements in our questionnaire.

The questionnaire (see Figure 8.1 for an abridged version and Appendix F.1 for the full questionnaire) is divided into five groups. These groups contain questions which

- collect general user information (1-12,29)
- collect task results (13,14)
- evaluate the motion behavior of the system (15-18)
- gather information about the usability of the system and application control and the usefulness of their features (19-26)
- gather information about the user's impression of the usefulness of guided exploration (27,28)

Questions 1-13 were posed after the first part of the task, 14-29 after the second part of the task.

Questions and Dependencies

The questions in the general user information part of the questionnaire were used to gather information about prior experience of virtual reality systems and interaction tools. A left-handed user (5) may have problems with using the joystick (right-handed). The learning type (6) may influence the type of answers in question 13 and 14. People with a difficulty to perceive colors (7) may only be able to use structural and time attributes to classify the artist's work, which uses color heavily. The evaluation of question 17 (dizziness) and 15, 16 (motion perception) depends on questions 8 and 9 (general tendency to motion sickness). The results of all questions that have to do with direct system control (interaction with the interface or menu), depend on the familiarity of the user with the interaction tool. It can not be expected that somebody who has never had any experience with a joystick or a stylus (or even a computer) instantly shows a good performance in controlling the system (19, 25) or finding features useful (23,24) which he or she may not be able to select.

Questions 13 and 14 are open questions to collect the knowledge gained about the pieces of artwork. A more classifying way of collecting this information would be to ask multiple choice questions about the features in the work, the names, the dates of creation. The disadvantage of this type of format, however, is that these question structure the answers of the users in a way they may not have thought of themselves, and, by this, trigger a post-learning process and a distorted answer. Multiple choice questions will also pre-structure the way the user looks at the second task and will also give different results there. Therefore, the question is formulated as an open text question. This is more difficult to evaluate but gives more accurate and interesting results. Question 29, concerning prior knowledge of the artists work, was asked to be able to classify unusually good performance in the open text questions in the right way. The rest of the questions do not depend on each other, and their content is self-explanatory.

Observer's Questionnaire

An observer was asked to observe the behavior, performance, and problems a user had while performing the task. The guiding but open text questions on the observers questionnaire were:

- How often does the user switch between self-navigation and use of the support system?
- How long is the user in each of these modes?
- How well does the user perform with the interaction tools and menu. Where are problems?
- Other observations not fitting in above categories.

Question		Type	1	4	7
1	name / ID	T			
2	age	T			
4	sex	CB			
5	which hand	CB			
6	learning type	CB			
7	difficulties to perceive colors?	CB			
8	general tendency to motion sickness	LS			
9	tendency to motion sickness in VE	LS			
10	which computer input devices known	CB/T			
11	virtual environments experience	CB/T/B			
12	which display systems known	CB			
13	what is remembered: names, colors and features of pictures	T			
14	what is remembered: names, colors and features of pictures	T			
15a	movement of camera: a. cameraposition	SD	To fast	just right	to slow
		SD	calm	just right	hectic
15b	movement of camera: b. rotation of camera	SD	To fast	just right	to slow
16	what if camera was interpolated (expert question: same movement but continuous)	SD	To fast	just right	to slow
		SD	calm	just right	hectic
17	feel dizzy when the supportive system	SD	Never	sometimes	always
18	time spent at each picture	SD	To short	just right	to long
19	judge the usage of the menu	SD/T	Easy		hard
20	did not have the control of the system	SD	Never	sometimes	always
21	system doing something you didn't intend it to	SD/T	Never	sometimes	always
22	system doing something you didn't expect it to	SD/T	Never	sometimes	always
23	is directly selecting and moving to a picture useful	SD	Never useful	sometimes	always useful
24	is filtering possibility (selecting attributes and period) useful	SD	Never useful	sometimes	always useful
25	start and stop the animation in an easy way	SD	Easy		hard
26	disoriented after using the supportive system	SD	Never	sometimes	always
27a	supportive system was useful a. in contrast to only navigating yourself	SD	Very useful		not useful at all
27b	b. in giving to you some ideas of what to look at	SD	Very useful		not useful at all
28	the supportive system made you gather more information and gave you a better understanding of the artists work	SD	Yes	at some points	no
29	What did you know about the Philip Gustons work before using the virtual museum	SD	Nothing	a little	a lot

Figure 8.1: Questionnaire. The two part usability questionnaire (part 1: 1-13, part 2: 14-29). The questions are shortened.

8.3.3 Participants

Ten participants, partly from the Virtual Environment group at Fraunhofer IMK, partly visitors, were asked to complete the user study. The users were all male, and their ages were between 21 and 38. Three of them had no or very little prior experience with virtual environment systems.

8.4 Results

As the conducted user study was meant to be informal and the number of participants was small, no statistic evaluation was done. The results are derived from the data (see Appendix F.2) and presented here in order of the corresponding hypothesis.

Hypothesis H1: All participants felt that the supportive system helped in gathering more information in a specific time frame. The fact that the users did not have to navigate themselves allowed them to observe the paintings while approaching them. This made them more relaxed and let them concentrate on the actual task. In comparing the real knowledge gained in the first and the second part of the study, we found that more specific features were recognized as more structural ideas are given through the interface. Groupings of pictures according to attributes and the period were also recognized. Some users, who had difficulties in using the interface, objectively did not learn much about the pictures themselves. These users still claimed, however, that the supportive system was useful to them. A special, additional training phase would have improved the performance of this group, as the overall time of five minutes appeared to be too short for learning to deal with the interface and doing the task itself. Overall, the results show support for hypothesis H1.

Hypothesis H2: Most users found it easy to start, stop, and restart the animation, which was done using a button on the joystick. This is a very crucial point in guided exploration as the users should be able to (re)gain control of the system at any time. Directly moving to a picture was mostly seen as useful by people who could well remember the names of pictures. One user actually used this feature to move automatically to distant but visible pieces of artwork. After selecting such a piece of artwork from the menu, some said that the actual movement was not necessary for them any more, as thumbnails are displayed in the menu. From these responses, we conclude that hypothesis H2 is verified.

Hypothesis H3: An important finding is that nobody felt disoriented after using the supportive system. This means that it is possible to use coexisting navigation and automatic motion in a virtual environment without confusing the users. Dizziness was only felt by one person who claimed to be easily motion sick. As disorientation and dizziness are suspected to reduce the sense of presence, the response to these questions indicate an approval of hypothesis H3.

Hypothesis H4: The movement of the camera was generally regarded as being too hectic. As explained before, we did not interpolate the results, therefore a new camera position was only adopted every 0.08 ms. This proves to be too slow for a smooth impression of movement. Nevertheless, most users did not feel dizzy when moved by the supportive system. The user's responses concerning the motion speed were on the complete spectrum from "too fast" to "too slow", which indicates that this strongly depends on user preferences. The system rarely did something that was not expected or intended.

When more than one piece of artwork was presented, some users claimed that the time spent at an individual picture was too short. Here, the users did not stop the system in order to view the artwork for the desired time, as was suggested before. The reason seems to be that the users felt comfortable being relieved of navigating themselves, wanting the system to take over completely. This may be because they adopt the passive behavior they are used to from linear forms of media like film and TV. These results indicate that hypothesis H4 is weakly verified, and the motion results should be smoothed.

Hypothesis H5: There were suggestions for improvements of the interface. Some users complained that the interface window was too big and in the way, even though it was transparent. Here again, the users did not dispose of the interface, which can be done with a single joystick button click. Another comment was that no feedback was given if no results were produced by querying the knowledge base or if all goals had been visited (stop of animation). Here, at least an audio feedback was requested. One complaint was that it is too hard to select an item in the interface and that there is no overview of all the selectable attributes. A last suggestion for improvement was to make information instantly available which is normally displayed on the label beside the piece of artwork. At some points, users had to change the menu to view the information after they used the supportive system. This meant clicking an additional time. The overall response to the interface (verbal and observed) was that it is difficult to handle. Therefore, hypothesis H5 does not hold.

8.5 Discussion and Summary

The CubicalPath system was applied to a virtual art museum in order to provide guided exploration to the users. Through user testing it was found that the aid can improve the performance of users in applications where they have to explore a virtual environment in order to gain knowledge. The supportive system also helped in giving guidelines on how to structure information in order to classify and to better remember this information. This is especially important to non-experts in the field of arts. Most users performed better when using the supportive system or at least welcomed this additional help. The concept of guided exploration with the system was rated as very helpful by most users.

It was found that the users could concentrate much more on the task itself if they were relieved of the navigation. This applied especially to users who were inexperienced in navigating with a joystick. The act of moving in the real world is done subconsciously and automatically. In a virtual environment, movement has to be initiated in an unusual way, and it is only an automatic process to experienced users. It was found that even experienced users could concentrate much better on the pieces of artwork themselves. In addition, some experienced users made use of the possibility of displaying the information of a certain artwork in view by employing the interface.

A framerate of 12 fps proved to be too slow for smooth animation. It did not confuse users, but interpolation of the data would probably improve the overall impression of the animation.

The crucial point when interacting with virtual environments is the interface. A system can be well designed, but if the interaction proves to be difficult, the system itself can not play its full strength. Interaction in immersive environments with a menu is not very intuitive. Therefore, these interfaces have to be enhanced, if systems introducing knowledge in the way the CubicalPath system does, should be controlled. A speech interface, for example, would let the users concentrate more on the tasks themselves.

It seemed that the users' feeling of presence in the environment did not decrease when they made selections with the interface and, more importantly, when the supportive system had control. This indicates that it is possible to use coexisting navigation and automatic motion in a virtual environment without confusing the users. Also, it was found that users do not lose their spatial context when using the CubicalPath system or the interaction tool.

This study does not use control groups, which would be important in more formal studies. It does not verify whether the knowledge gain in the second task is a result of the extended time spent in the exhibition or whether the supportive system was responsible for this. The resulting answers in question 14 indicate the latter, but these results may also be achieved by simply showing the user the list of attributes and then letting them again self-explore the museum.

No user seemed to have problems with understanding the task or the application. We think that with this test, the focus was on the method rather than on the application.

9 Concluding Remarks

Exploration in a virtual environment is a highly interactive process which involves constant decision making and travelling for the purpose of discovery. Travel, the process of moving through the environment from the current position to a single target, is one of the major activities in virtual environments and the main interactive task for exploration. Travel in virtual environments is typically done by direct user control of the view, which requires considerable skill to control the interaction. For this, it is necessary that the position of the target in the virtual environment be known to the user, which may not apply to visually hidden targets or non-visual targets like sound and olfactory sources. Targets in the context of exploration may only be recognized as such, once they are already in full view.

9.1 Discussion

The purpose of this thesis was to find a method to support exploration in virtual environments. This can be done both on the cognitive level of target decision making as well as on the level of travelling to identified targets. We support decision making by integrating the output of common queriable information spaces into our system. We support travelling by automatically moving the user inside the resulting goal field, presenting the identified targets one-by-one. At the same time, we give, if requested, full interactive control over the motion, or we overlay user input with the system output. By doing this, we give the user the feeling of being in full control of the system. The user has also the possibility of letting the system do the travelling, thus relieve the user from this task and instead let her focus on exploration.

We developed our system in two steps. First, we introduced a new, real-time capable approach to automated travel in dynamic, unpredictable virtual environments based on the potential field method. It is derived from the physics of the motion of a charged particle in an electric potential field. This method uses a discretized representation of the environment in a uniform rectangular grid. Automated travel in virtual environments is technically equivalent to automated view or camera motion. In our method, the target attracts the camera while obstacles repulse it. The potential field method is a local method that does not require planning. Avoiding expensive planning is necessary because, in a dynamic environment, the target and the obstacles may continuously be repositioned. Nevertheless, in cluttered, maze-like environments, this local method may result in the camera moving into an unwanted local minimum of the potential field. In this case, we rely on the user to move out of this situation by using the generally in virtual environments available interaction tools for navigation. This situation rarely occurs in sparse environments. We developed and implemented a system-independent, auxiliary supportive system to automate travel.

In the second step, we extended the automated travel system to be able to move in a goal field, a field of multiple targets generated from the results of a goal-directed query to an application information space. Each target can be qualified with a different level of relevancy, the attractivity to the user in the context of the goal.

For this, we introduced the dynamic potential field method. This method deals with multiple targets by adjusting the attractivity of visible targets. The targets are presented one-by-one, ordered with respect to the attractivity and the distance of the targets from the current camera position. The view of the camera is constantly evaluated to gradually decrease the attractivity of visible targets. This, in effect, releases the camera from visited targets after a while. The target's importance influences the duration of stay in front of the target. By adjusting the attractivity, the history of visited targets is stored in the goal field and these targets will not attract the camera any more. The camera is oriented in a way that meaningful content – a target – is always in the view, if possible. This decouples the orientation of the camera from the direction of motion.

The dynamic potential field method results in a system which is able to deal with interactive, real-time input by the client application or the user. The input can be dynamic, unpredictable object locations, which influence the geometric setup; dynamic or re-adjusted targets, which influence the attractivity setup; or a dynamic view, which assigns a new current camera view to the motion generation function. It is also possible to influence the results of the motion generation function with an additional force input generated by an interaction tool.

The core motion generation method was extended to include some possibilities for a specific presentation or incorporation of global knowledge. We introduced the concept of *navigation objects*, additional, often invisible targets or repulsive objects. Navigation objects allow to introduce global knowledge into the environment by enabling the algorithm to follow a complicated path or by preventing the camera from moving into known dead ends. They also make director-specified views possible: for example, viewing objects from a certain perspective for dramaturgical reasons. In order to make this possible, the targets for the position and orientation of the camera of the camera, were separated. Navigation objects represent spatial, high-level information about the scene, and can also be used to include non-visible targets, like auditory, olfactory or tactile targets, in virtual environments.

The geometric setup of our system requires a real-time capable voxelization method. We presented two suitable voxelization methods for spatial analysis, a software-based and a hardware-based method. Both methods generate object-coded volume data (each volume element is assigned a list of occupying objects) from geometric surface data by utilizing the Performer scene-graph. Surface data is used as interior of geometry objects is not visible, and thus it makes no sense to include them into the calculation. In the software-based method, a 3D scan conversion based triangle voxelizer, each object's polygons are transformed into the voxel space. This makes the transformation an "object to voxel" approach, as for each object the occupying voxels are identified. The hardware-based method is a "voxel to object" method, which renders and analyzes the complete voxel space to retrieve the objects inside each voxel. A special approach to color-coding allows the identification of multiple objects per voxel, which then gives the same results as "object to voxel" approaches. The performance measurements show that the software-based method is preferred for its simplicity and its real-time behavior if only a small number of objects are dynamic, while the hardware-based approach shows its strength if the *z resolution* – the height of the space – is small compared to the x and *y resolution* – the large area that can be navigated.

The CubicalPath system is designed as an auxiliary supportive system. It uses a platform and machine independent client-server architecture based on CORBA. The virtual environment application and the application information space are connected to the server system through a lean interface which mirrors their common data structures: the scene-graph, the transformation matrix connected to the application viewer, and a list of object data resulting from a query to an information space. Camera data is continuously generated by the system and transmitted to the client application, where it is connected to the transformation matrix of the client application viewer. This results in the CubicalPath system being easily integrated into existing virtual environment applications in the same way as an additional interaction tool would be integrated.

We successfully proved our concept and system with three applications. The first is a medical training system, run on a desktop computer, in which the camera browses through an object-cluttered space with 5DOF (up-direction fixed) and the objects can be repositioned any time. The second application is the virtual art museum, run in a CAVE-based system, in which the camera moves at a fixed height above the museum floor with 4DOF. This application shows pieces of artwork on the wall from a director specified viewpoint. The third application is the Bonn Marktplatz application, run in the i-Cone, in which dynamic objects move through a space with many obstacles. The camera moves in the same way as in the previous application.

In an informal usability study with ten users carried out on the virtual art museum application, we evaluated different aspects of our approach to support exploration. The usability study showed that guided exploration can improve the performance of users in applications in which they have to explore a virtual environment in order to gain knowledge. The supportive system also helped in giving guidelines on how to structure information in order to classify and to better remember this information. Most users felt they performed better, or at least welcomed the additional support. The concept of guided exploration with the aid of the system was rated as very helpful by most users. It was also found that the users could concentrate much more on the exploration task itself if they were relieved of the travel task. This applied especially to users who were unexperienced in navigating with a joystick. Even experienced users found that they could concentrate more on the pieces of artwork, when the support system was used.

The study showed that considerable research is necessary to improve interfaces to system control. It was found that the desktop-like window interface (see Figure 7.5) which we use as the access point to the application information space is tedious to use in virtual environments. We found that the users' feeling of presence in the environment did not decrease when the supportive system was in control. This indicates that it is possible to use coexisting navigation and automated travel in a virtual environment without confusing the users.

9.2 Future Work

This system can be seen as a start in work on presenting content in virtual environments. There exist some real-time or on-the-fly methods for presentation on desktop applications which incorporate cinematographic rules for presentation [TBN00]. These rules have to be reconsidered, adapted, applied, and evaluated for immersive virtual environments before systems which support the new "VE-tographic" rules, can be built.

Up to now, the parameters controlling the motion behavior of the CubicalPath system have been modified by hand if they did not match the application in the desired way. It would be sensible to develop an automatic tuning system which analyzes the geometric environment (for example its density) and adjusts the field functions and control parameters to match to a specified behavior description.

The issue of unwanted local minima, the main drawback of the potential field approach, should be investigated. We and others have proposed methods like automatically filling concave objects, introducing global knowledge by navigation objects, using Brownian motion, or using a search to move out of the minimum. These are methods which are applied when the camera already stopped in a minimum. More desirable would be a method, which prevents the movement into these minima in the first place or which prepares the field in a way, such that it has only minima at positions where a target is defined.

The CubicalPath system, as it is now, works best in sparse environments with convex objects; Because of unwanted local minima, it will fail in complex environments like mazes. A combination of the CubicalPath system with a set of pre-calculated pathes may provide for an efficient solution to real-time supported exploration even in dynamic complex environments, so long as the complex part of the environment itself is static.

For generating the goal field, improved information spaces or knowledge bases need to be considered or developed which focus on supplying context-relevant information. A starting point was the knowledge base integrated in the 3D puzzle which generates semantic context.

Generally, a lot of work still needs to be done in the area of user interfaces to access data in virtual environments. As long as interfaces to textural and numeric system control are difficult to use, interactive applications which rely on these interfaces may not evolve. New solutions have to be found for the intuitive and easy representation and manipulation of such data in virtual environments.

For large resolutions of the cube space, the camera data have to be interpolated as then, the response time of the system may not provide for a smooth motion of the camera. This interpolation should be C^2 continuous and should not require too many steps in advance for the calculation. The more steps in advance needed, the longer is the response time of the system. This may result in a failure of the collision avoidance, as dynamic objects may have moved into the path which was calculated for some previous time step based on the geometric data at that time.

A Mathematical Derivations

The electrical field of a point charge is the derivative of V_{point_charge}. If U is only dependent on $r = |\overrightarrow{r}|$ then grad $U(r) = U'(r)\frac{\overrightarrow{r}}{r}$ (derivation for a central symmetric field [BS91]). Here, the full derivation of Equation A.1 with Cartesian coordinates is given. Exemplarily, in Equation A.2 the partial derivative for x is developed. This is then used in A.3 to calculate $\overrightarrow{\epsilon}_{point\_charge}$(see A.4 and A.5).

$$\begin{aligned}
\overrightarrow{r} &= \begin{pmatrix} x \\ y \\ z \end{pmatrix} \\
V_{point\_charge}(\overrightarrow{p}) &= k\frac{q}{|\overrightarrow{p} - \overrightarrow{q}|} = k\frac{q}{\overrightarrow{r}} \\
\overrightarrow{\epsilon}_{point\_charge}(\overrightarrow{p}) &= \overrightarrow{\nabla} V(\overrightarrow{p}) = \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix} V(\overrightarrow{p}) = \begin{pmatrix} \frac{\partial V(\overrightarrow{p})}{\partial x} \\ \frac{\partial V(\overrightarrow{p})}{\partial y} \\ \frac{\partial V(\overrightarrow{p})}{\partial z} \end{pmatrix} \qquad (A.1)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial V(\overrightarrow{p})}{\partial x} &= \frac{\partial k\frac{q}{|\overrightarrow{r}|}}{\partial x} \\
&= \frac{\partial k\frac{q}{\sqrt{x^2+y^2+z^2}}}{\partial x} \\
&= kq\frac{\partial}{\partial x}(x^2 + y^2 + z^2)^{-\frac{1}{2}} \\
&= kq2x(-\frac{1}{2})(x^2 + y^2 + z^2)^{-\frac{3}{2}} \\
&= -kqx(\frac{1}{\sqrt{x^2 + y^2 + z^2}})^3 \\
&= -kx\frac{q}{|\overrightarrow{r}|^3} \qquad (A.2)
\end{aligned}$$

$$\overrightarrow{\epsilon}_{point\_charge}(\overrightarrow{p}) = -\begin{pmatrix} kx\frac{q}{|\overrightarrow{r}|^3} \\ ky\frac{q}{|\overrightarrow{r}|^3} \\ kz\frac{q}{|\overrightarrow{r}|^3} \end{pmatrix} \tag{A.3}$$

$$= -k\frac{q}{|\overrightarrow{r}|^3}\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$= -k\frac{q\overrightarrow{r}}{|\overrightarrow{r}|^3} \tag{A.4}$$

$$= -k\frac{q}{|\overrightarrow{r}|^2}\frac{\overrightarrow{r}}{|\overrightarrow{r}|} \tag{A.5}$$

B Implementation Details

B.1 Extended 3D Line Voxelization

The function `cpDCS::voxelizeLine` on the following page voxelizes one polygon described by the points $p1$, $p2$ and $p3$ and returns the number of found voxels. The flag *checkP3* indicates, if the line from the current voxel to $p3$ has to be computed. First the line between $p1$ and $p2$ is computed. Each found voxel is set to be the new point $p1$ and point $p2$ is set to be $p3$. By this the line between the current voxel and $p3$ is also computed. The algorithm is adapted from Graphics Gems IV - Voxel traversal along a 3D line [Coh94]. It assumes that the line endpoints lie on an integer grid.

```
int cpDCS::voxelizeLine (pfVec3 p1, pfVec3 p2, pfVec3 p3,
                         bool checkP3){

    int n, sx, sy, sz, exy, exz, ezy,
    int ax, ay, az, bx, by, bz;
    int dx, dy, dz, x, y, z;
    int noOfHits = 0;
    dx = p2[0] - p1[0];
    dy = p2[1] - p1[1];
    dz = p2[2] - p1[2];
    x = p1[0]; y=p1[1]; z=p1[2];

    sx = sgn(dx); sy = sgn(dy); sz=sgn(dz);
    ax = abs(dx); ay=abs(dy);   az=abs(dz);
    bx=2*ax;      by=2*ay;      bz=2*az;
    exy = ay-ax;  exz=az-ax;    ezy=ay-az;
    n=ax+ay+az;

    if (n== 0)
      noOfHits += VisitVoxel(x,y,z);
        else {
          while (n--){
            voxelizeLineUsed = true;
            //Visit the Voxel
            noOfHits += VisitVoxel(x,y,z);
            if (checkP3)
              noOfHits += voxelizeLine( pfVec3(x,y,z), p3, p3, false);
            if(exy<0){
              if (exz < 0){
                x+= sx;
                exy += by; exz+=bz;
              }
              else {
                z+=sz;
                exz -= bx; ezy += by;
              }
            }
            else {
              if (ezy <0) {
                z+= sz;
                exz -= bx; ezy +=by;
              }
              else{
                y += sy;
                exy -= bx; ezy -= bz;
        } } } }
        return noOfHits;
}
```

B.2 IDL File: Interface Description

The IDL file is the basis for the CORBA environment to build a skeleton class which then has to be implemented by the server applications. It contains the definition of data structures (B.2.1) used by the interfaces and the description of the interfaces themselves. The application interface iCPApplication (B.2.2) has to be implemented by the client application and is used by the CPServer. The CPServer interface description, iCPServer (B.2.3), illustrates the commands which are used by the client application to control the CubicalPath system. The CPAnalysisServer interface (B.2.4) is used for the communication between CPServer and CPAnalysisServer.

B.2.1 Definitions

```
//=== Structs and Definitions
struct CPVector{
        double x;
        double y;
        double z;
};

struct CPTriangle{
        CPVector Tri[2];
};

struct CPCameraData{
        CPVector pos;
        CPVector dir;
        CPVector up;
        CPVector center;
        long ID;
};

struct PosStruct{
  unsigned long x;
  unsigned long y;
  unsigned long z;
  float count;
};

typedef sequence<PosStruct> CPViewedPositions; typedef
sequence<CPVector> CPTris; typedef double CPMatrix[16];
```

B.2.2 Application Interface

```
// =================== iCPApplication ===================
//the Interface between the Applikation and the CPServer
interface iCPApplication {
//receive a connection ok. Server is ready and listening to this Application
  void serverListens();

//===== data receiving functions for client sent by agent
//receiving Cameradata
  oneway void set_Camera(in CPCameraData cam);

//receiving CubeSpace data for visualization or reference
  oneway void set_CubesData(in CPViewedPositions newCubes);
}; //end of iCPApplication
```

B.2.3 CPServer Interface

```
//====================== iCPServer ======================
interface iCPServer {
//===== controlling functions from App to agent
//start iteration  (no of iterations)
  oneway void runIteration(in long anzIterations);

//stop iterations
  oneway void stop();

//===== maintenance of ID values
//delete ID
  oneway void deleteID();

//delete everything and reset agent
  oneway void reset();

//setting controlling function to influence behaviour of Server
  oneway void set_controlFunction(in string controlFunction);

//===== data sending functions from app to agent
//*** Initialisation
//initialisation of CPServer with Corbahandle and Size
  oneway void newSession(in string objRef);

//delete the current session. reset of CPServer
  void deleteSession(in string objRef);

//sending cubesspace resolution (no of cubes x, y, z)
  void set_CPsize(in long sizeX,in long sizeY, in long sizeZ );
```

```
//sending Bounding Box of Szene
//  (the space in which the camera will move in Szene coordinates)
  void set_BBScene(in CPVector min, in CPVector max);

//==== geometry
//sending ID and TransformationMatrix
  oneway void set_Transformation(in unsigned long ID, in CPMatrix m);

//sending complete Voxelrepresentations stored in file
  oneway void set_DataFilename(in string filename);

//sending ID and filename
  void set_Geo_File(in unsigned long ID, in string filename,
                    in CPVector center, in CPVector size);

//sending ID and List of Triangles (3 vertices * no of Tri)
  void set_Geo_Tris(in unsigned long ID, in CPTris Tris);

//sending ID and List of Triangles (3 vertices * no of Tri)
//and OBJECT, GOALVIEW, GOALPOSITION
  void set_Geo_Tris_Goal(in unsigned long ID, in CPTris Tris,
                         in string kindOfObj);

//sending ID and List of Cubes. Values are encoded as Integers in CPTris
//the prior two functions will be used. The mode (hardwarebased for
//VoxelizationMethod ) is set by send_command_withValue(...

//connection between Objects and goals - views and/or position
  void connect_Obj_Goals(in unsigned long IDObj, in unsigned long IDPosition,
                         in unsigned long IDView);

//deletion of Geometry belonging to ID
  void delete_Geo(in unsigned long ID);

//clearing of all geometry (reset)
  void reset_Geo();

//*** I values  (attraction values)
//sending ID and I value
  oneway void set_I(in unsigned long ID, in double attractionValue);

//*** Cameraposition and values
//sending cameraposition and direction
  oneway void set_Camera(in CPCameraData cam);

//sending cameraposition and center
  oneway void set_CameraCenter(in CPCameraData cam);
//requesting actual Cameraposition
  CPCameraData get_Camera();

//==== General controlling function
//a string is sent to trigger an attribute in the control unit
  oneway void send_command(in string command);

//a function name and attribute is sent to the control unit
  oneway void send_command_withValue(in string command, in double Value);
};
```

B.2.4 CPAnalysisServer Interface

```
//======= Interface between CPServer and CPAnalysisServer ======
interface iCPAnalysis {
//initialization of CPAnalysisServer
  oneway void initCPVA();

//sending of camera data
  oneway void set_Camera(in CPCameraData cam);

//collecting camera data
  CPCameraData get_Camera();

//triggering of analysis of camera data. receive results
  CPViewedPositions analyseViewedData(in CPCameraData cam);

//sending cubes
  oneway void set_CubesData(in CPViewedPositions newCubes);
};
```

C Imaging Systems

Cameras for photo and film are imaging systems. They map the real world, the scene, into the image plane (see Figure C.2). The process of imaging can result in one image, a photo (more precisely its negative), or a sequence of images, a video or a film.

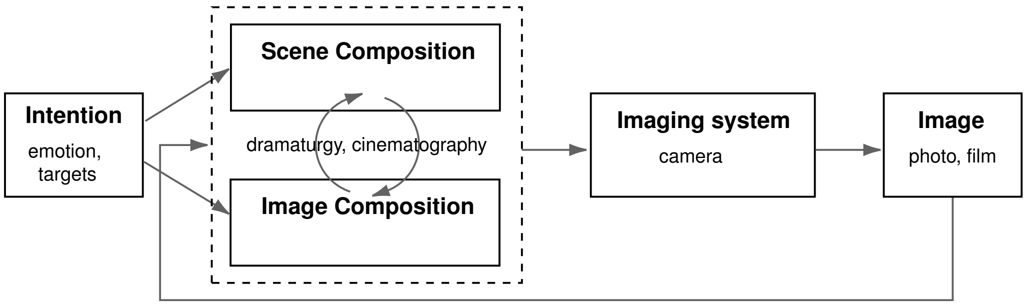


Figure C.1: The Process of Imaging.

There are several steps involved in the process of imaging, illustrated in Figure C.1. First, the *intention* is defined. The intention may for example be to visualize a scene, to tell a story, or to create emotions with the piece, image, or film. Second, based on the intention, the *scene composition* and the *image composition* is fixed. The scene composition describes the position and orientation of actors and objects in the scene. The image composition describes where on the image the actors and objects should appear. Third, the image is created by adjusting the attributes of the camera – the *imaging system* – and taking the picture or recording the film. The scene is mapped into the image space, showing the desired image composition. All steps depend on each other and often results of a latter stage iteratively re-influence a former stage.

A film adds the dimension *time* to the images. In a film, the scene's composition, the image composition, and the parameters of the imaging system can change over time. The camera may stand still or move in different ways to create a particular atmosphere or to visualize a certain situation or action. It may change its height relative to objects or jitter. This allows for a large set of dramaturgical elements described in the art of cinematography [Ari76, Kat91], which are established for film making. These can also be important for computer-generated animations as cinematographic rules in real films established expectations of spectators of computer generated animations.

Figure C.2 illustrates the task of an imaging system, here a camera. Generally, an imaging system projects an n-dimensional space – the object space – into an m-dimensional space – the image space – by using a projection function. n can be larger, equal or smaller than m. In the case of a real or virtual camera in a 3D environment, n is 3, m is 2 and the parameters of the imaging system, the camera and the camera placement, form the projection function. This projection function is set up to produce the image of the world with the desired composition.

A virtual camera takes an image of the virtual object space as the real camera takes an image of the real world. This work is about virtual camera motion. Nevertheless, the setup of virtual cameras borrows largely from real cameras and can easily be defined in terms which are defined and established for real cameras. The following sections discuss the imaging process of real cameras which can be seen as foundations for describing virtual camera representations in Appendix D. This is done for a static photo camera, because it has similar technical features as a film camera. The difference is that a film camera continuously generates new images, instead of taking only one.

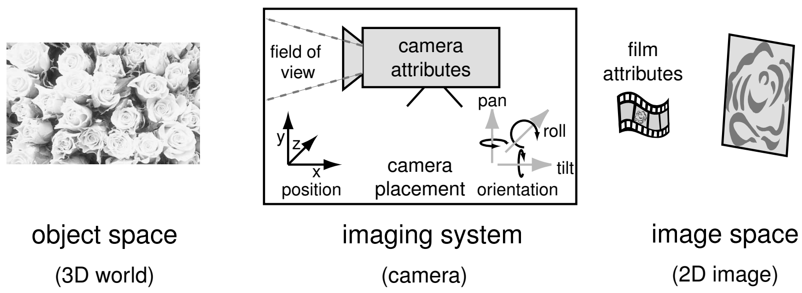


Figure C.2: Imaging System. A photo or film camera – an imaging system – maps the 3D world into a 2D image. A camera has internal mapping attributes like the field of view, the aperture value, the shutter speed. The parameter for the camera placement describe the camera's position and orientation in the world. Together they form the parameter of the imaging system and define the 2D image for a given 3D world. The film attributes define which of the information mapped in the image space will be stored on film and therefore displayed in the final image.

C.1 Camera Attributes

A real camera consists of a lens, an aperture, and a shutter. They control several parameters – the camera attributes listed in Figure C.3 *A* – which influence how the *real world* is projected into the *image space*, thus on the film plane.

A **lens** is defined by its *focal length*. The focal length of a camera lens is the distance from the film plane of the camera to the center of the lens when the lens is focused on infinity. It defines the *zoom factor*: the longer the focal length, the narrower the angle of view and the more magnified the image of an object at fixed distance is. By this, it also defines the *field of view*, how much of the world appears on the image.

The **aperture** controls the size of a hole, usually located at the base of the lens. The aperture value is the ratio of the focal length of the lens to the diameter of the hole. The aperture also controls the *depth of field*. The depth of field can be thought of as the amount of the image which has acceptable sharpness. The smaller the aperture value, the larger the depth of field. A point size aperture (a pin-hole camera) displays all objects, near and far, sharp.

The **shutter** protects the film from exposure. The *shutter speed* is a measure of how long the shutter remains open when the picture is taken, thus how long the film is exposed to light.

By adjusting the aperture and shutter settings, the way light and film interact is changed. The wider the aperture and the longer the exposure, the more light falls on the film plane. The *image size* on the film plane, the area on which the object space is mapped, is fixed for the camera. All other components are adjusted to reproduce an images of this size and aspect ratio.

A: Camera Attributes

depth of field	depth of sharp area
field of view	wide or narrow angle
focal length	zoom factor
shutter speed	exposure time
image size	the area on the film plane exposed by light

B: Placement Attributes

position	x, y, z of the position
pan	the rotation angle around the vertical axis (y-axis)
roll	the rotation around the axis which forms the viewing direction
tilt	the rotation angle around the axis perpendicular to the viewing direction (x-axis). The camera may look upwards or downwards instead of horizontal and parallel to the floor.

C: Image Attributes

image size	defines the height and width of the image
granularity	defines the resolution of the image
sensitivity	defines how sensitive it reacts to light
color space	b/w or color image

Figure C.3: Camera, Placement, and Image Attributes.

C.2 Camera Placement

Setting up a camera in a physical place allows for six degrees of freedom (6 DOF), three for position and three for orientation. A camera is fixed at a 3D location and it can be rotated around three axes like in Figure C.2. The terms for camera orientation are pan, roll and tilt as in Figure C.3 B. When panning a camera, the horizon stays parallel to the image[1]. Rolling tilts the horizon relative to the image's bottom line. Tilting changes the perspective to frog or birds-eye perspective.

C.3 Image Attributes

The receiving medium – the **film** – has its own set of attributes listed in Figure C.3 C. The film attributes define which of the information mapped in the image space will be stored on film and therefore displayed in the final image. A film has an image size, a granularity, a sensitivity and a color space. The *size of the image* is adjusted to the camera's image size. The *granularity* defines the resolution of the image. The larger the number of grains in the film, thus, the smaller a grain itself, the more details can be resolved on it. A high *sensitivity* allows short shutter times for high-speed photography. And a film's *color space* is either designed for b/w photography or for color images.

Photography is associated with the final reproduction of the image on paper. This reproduction is a post-production step which is independent of the imaging process discussed before, but relies completely on the image on the film. This step has its own set of attributes and possibilities but it can not backwards influence the image on the film. The same holds for movies. Showing a movie is simply enlarging the image on the film. Therefore, the image on the film is regarded the final image of the imaging process.

[1]In a single image with a visible horizon panning is not noticeable. It is the same, if the tripod is rotated by 90° or the camera on the tripod is panned by 90°.

D View specification

D.1 General View Specification

A virtual camera maps points of the 3D virtual world into the 2D image space. It is an abstraction of a real camera. It is also an imaging system which maps the virtual object space in the image space – here the view plane – and eventually on the screen. Imaging systems, as in Figure C.2, and real camera attributes were discussed in Appendix C.

In computer graphics, the term viewer or view is used synonymously to the imaging system parameter, thus the camera. The view can be specified by the *view position, view direction,* and the *view plane* as in Figure D.1. They form the projection function from 3D to 2D. The final image additionally depends on the the *view frustum* and the viewport specifications which are the *image/window size, resolution*, and *position* relative to the view plane. The view frustum for a perspective view is a truncated pyramid which is centered at the direction of view and is bordered by the near and far clipping planes.

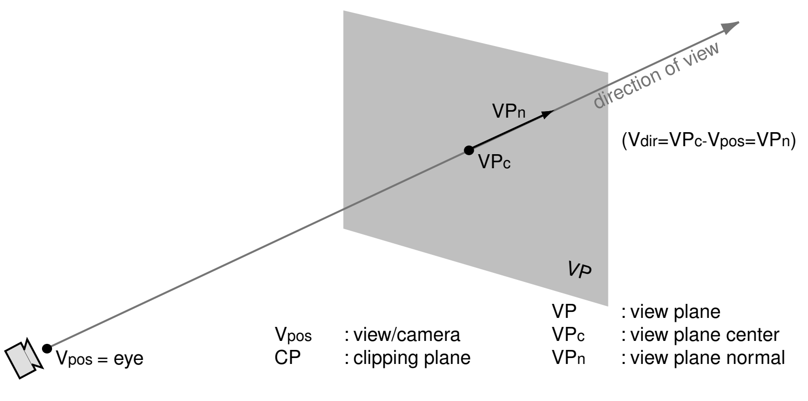


Figure D.1: View Plane and View Plane Normal. The view plane can be independent of the near clipping plane.

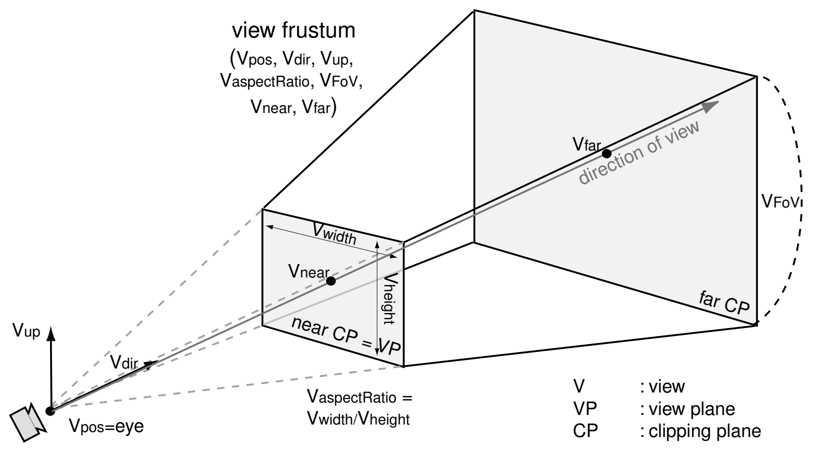


Figure D.2: OpenGL Perspective View. The view is defined by the view point, the view frustum and the view plane size and resolution. For OpenGL the view plane is equal to the near clipping plane.

A virtual camera can completely simulate a real camera. With ray-tracing any kind of lens system in combination with aperture and shutter speed settings can be simulated to create photo-realistic images with depth of field, motion blur, lens flare, shadows and such [KMH95]. Because ray-tracing is a time consuming process, real-time rendering systems like OpenGL$^{\text{TM}}$and Performer$^{\text{TM}}$use the pin-hole camera as a model. This camera has an infinitesimal small aperture, thus, the complete object space in the viewing frustum is rendered in focus[1]. For virtual environments, typically, the pin-hole camera model is used, as the purpose is to visualize the 3D space in stereo and in real-time, not to render photorealistic images[2].

There are multiple ways to specify a perspective view. The first way, illustrated in Figure D.1, specifies the view plane (VP) independently from the view position. The view plane normal (VPn) is equal to the direction of view (Vdir). In OpenGL this is the negative z-direction. The direction of view is either directly specified by Vdir. Then the center of the view plane (VPc) is calculated. Or, like in this illustration, Vdir is derived from VPc (Vdir = VPc - Vpos).

The second way to specify a perspective view, illustrated in Figure D.2, is based on the OpenGL camera. A view coordinate system is established with the camera positioned in the origin and pointing in the direction of view (Vdir). An up-vector (Vup) establishes the up direction of the camera and the view plane (VP). The view

[1]Also with a pin-hole camera model, the sharpness as a function of the depth of field can be simulated by rendering and blending multiple images from different view positions. This again adds a large overhead (multiple renderings) for one image.

[2]Unless the users current real eye convergence is measured, the complete scene has to be rendered in focus, as it is not known, if the user focuses on near or far parts of the scene.

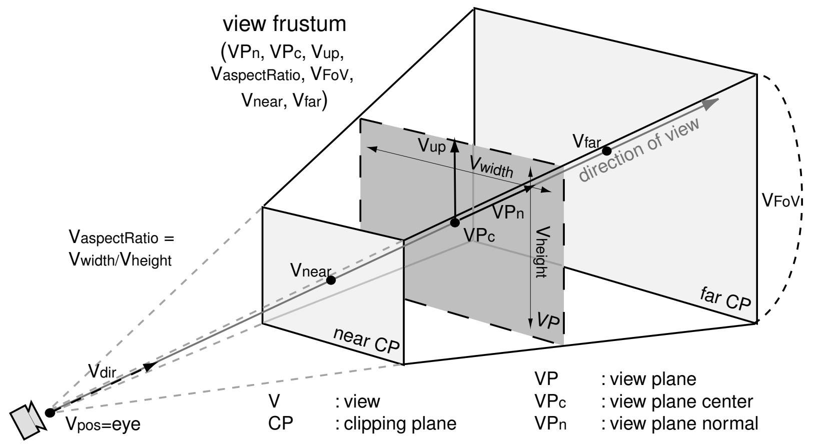


Figure D.3: View Plane and Clipping Planes.

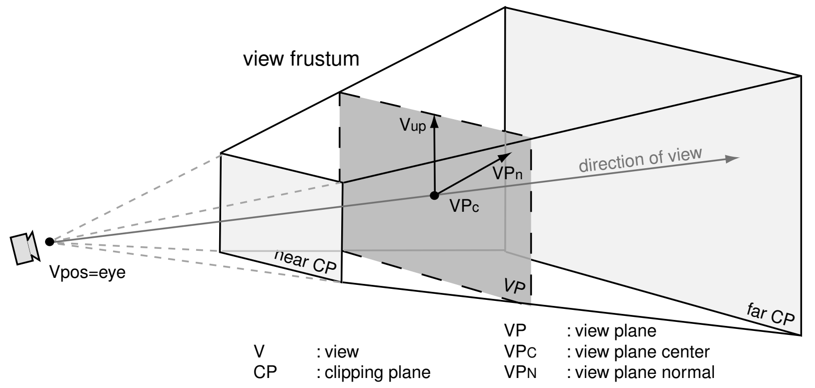


Figure D.4: View Plane Normal Unequal Direction of View. VCdir is defined as VCpos - VPC. It is not depended on the users view direction.

frustum is defined by the field of view (Vfov), the aspect ratio, and the near and far clipping planes. In OpenGL, the view plane is set to be the near clipping plane. The size of the view plane depends on the field of view and the aspect ratio. The aspect ratio specifies the width relative to the height of the clipping planes, and by this also the view plane.

Generally, the **view frustum** specifies which objects of the 3D scene are rendered. It selects the *field of view*, but also clips objects in a distance larger than the far distance value (Vfar)[3]. The definition of the view frustum also influences the *visibility of objects*, thus which ones may be occluded and which ones are likely to be visible. The latter case holds for close objects. Also, because of the perspective view, *close objects appear larger* relative to distant ones.

Figure D.3 shows the same setup as in Figure D.1 but also specifies the view frustum. By this, it defines the borders of the view plane, the near and the far clipping plane.

[3]Different to real cameras, rendering systems need to utilize a depth buffer for correct ordering of object's pixels in the image depending on their distance to the viewer. This buffer can not be arbitrarily large and highly resolving. Therefore, the depth of the rendered space is bound to useful values, normally to the scenes borders.

They are defined by the near and far distance from the view position (Vpos). Their size again depends on Vfov and VaspectRatio. The specifications in this illustration form the same view as in Figure D.2 using different and more general parameter. This can be seen in the Figure D.4, where the VPn is different to the direction of view. This is an important case in stereo systems and in static head-tracked virtual environment installations as will be described shortly.

D.2 View Specification and View Motion in Virtual Environments

Head-tracked virtual environment system rely on the user selecting his or her own position and direction of view in the available physical space. The available space for this is called the view platform. The view platform forms all possible physical positions the user can adopt in the virtual environment system. This physical space is often bound by walls, if used indoor, or by the display system itself. The view platform is illustrated in Figure D.5 – here it is a view box – for a CAVE [4].

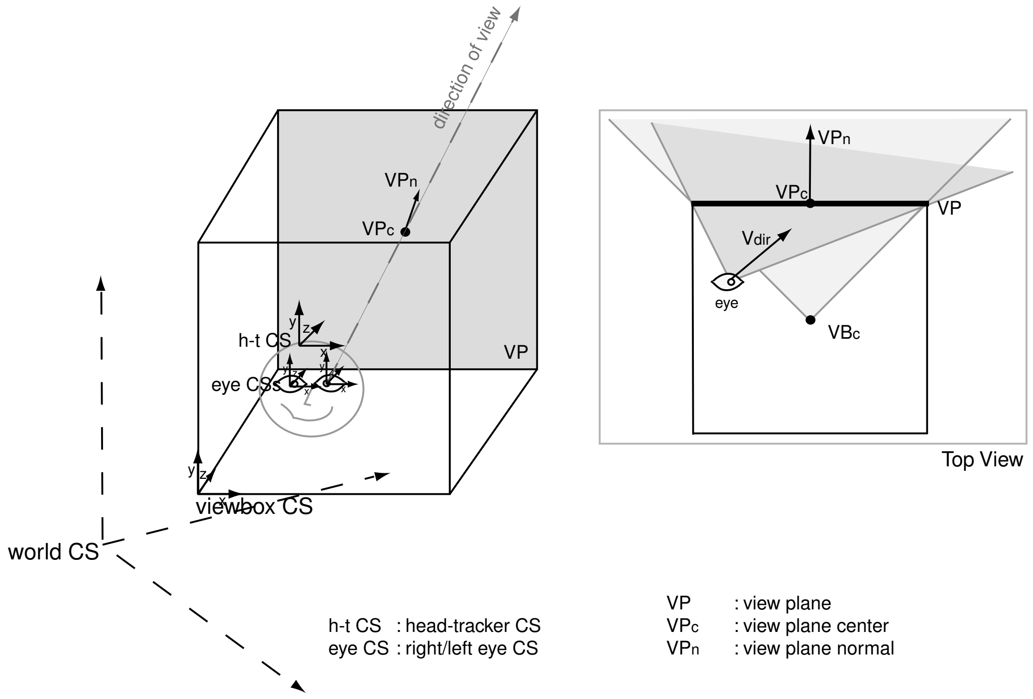


Figure D.5: View Platform. The view platform for the CAVE is a box, here called viewbox.

[4]The CAVE setup is illustrated in Figure E.2

View Setup/Parameter

A view is rendered depending on the view position and the view planes. The position and orientation of the view plane(s) in projection-based virtual environment system is fixed as the display's screen is equal to the view plane. Head-mounted displays (HMD), for example, have fixed screens relative to the eye. Thus, the view plane is perpendicular to the direction of view and the view setup is as in Figures D.2 and D.3. The user selects a part of the 3D world by orienting his or her head and only the view in this direction is rendered.

In fully immersive large scale installations, like the 6-sided CAVE, the complete 360° space is rendered. Different to a HMD, in a CAVE the screens are fixed in the real world. This fixes also the view plane coordinate system relative to the real world. Now, when moving the user's eye position inside the view platform, the view frustum gets distorted as illustrated in Figure D.4. Only when the user's head is in the middle of the view platform– the VBc –, the undistorted perspective view is rendered. The further the user moves into one corner of the system, the more distorted the view gets, as the 3D world is projected on the 2D screen which itself is viewed from the side, not the front(see Figure D.5 (Top View)). This feature requires tracking the user's head and it is called off-axis projection.

View Motion and Coordinate Systems

In a system like a CAVE, it is not possible to explore a larger world than the display system solely by walking. Even if it is possible, the speed of walking may be slow, if large distances are to be overcome. For this, interaction tools are utilized, which move the view platform relative to the 3D world [PST$^+$96].

This results in two possible types of motion in a virtual environment system: the first is the motion of the view platform through the environment. The second is the motion of the user inside this view platform. Both types of motion can occur simultaneously. The combined result forms the view position $Vpos$ of the view to be rendered.

In the case of the CyberStage, Fraunhofer IMK's four sided CAVE, motion of the view platform is equal to moving the CyberStage through the virtual 3D world. Independently from the user position, the virtual CyberStage coordinate system, the view platform, is moved relative to the 3D world.

The user's eye position inside the view platform is derived from the head-tracker output. The head-tracker itself provides coordinates relative to the physical Cyber-Stage, because it observes the physical system space. The user's eyes additionally have a fixed offset to the head-tracker. Figure D.6 shows the dependencies of these coordinate systems (CS). For generating a view – a specific eye/screen combination –, the eye position and the view plane (the screen) have to be transformed into the common view coordinate system, for example the 3D world CS.

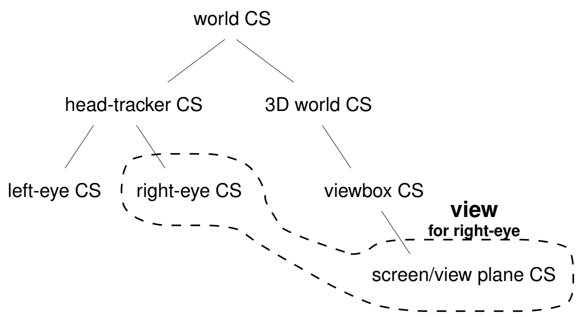


Figure D.6: Coordinate System Dependencies for Head-Tracked Stereo Views.

Changes in the head position and orientation will modify the eyes' position in the world CS. Nevertheless, the resulting eye direction does not influence the rendering of the view. Only the eye's position is a parameter to the projection function. Therefore, the user is still free to view in any direction without modifying the scene relative to the screens (except small jitter movements introduced by the head movement).

Fully immersive systems like HMDs provide no preferential direction of view. When moving the view platform with some interaction tool this is usually done relative to the current users direction of view. In the CAVE, users tend to position themselves in front of one of the screens. For IMKs four sided CyberStage this is normally the front screen, opposite to the entrance. With this knowledge, it is possible to derive a common direction of view, which is required for automatic view motion for presentation.

Adding user or application controlled motion to the scene means moving the view platform. The output matrix of an interaction tool is attached to the position and orientation of the center of e.g. the CAVE, the view platform. Independent of this movement the user can additionally move on this platform and change the eye position as explained before.

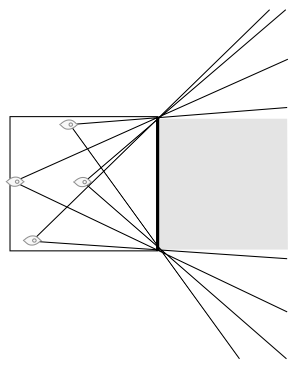


Figure D.7: Eye Position in View Platform Influences View Frustum. The eye position influences the direction and opening angle of the view frustum. There is one box-shaped area (grey) directly behind the screen which is always visible, independent from the eye position of the user.

E Virtual Environment Systems

E.1 Responsive Workbench

The Responsive Workbench (RWB) [KF94], developed at GMD (now Fraunhofer IMK) in 1993, provides a human-machine interface modelled after people working on desks, workbenches, and tables. The objects, displayed as computer generated stereo images, are back-projected onto one or two screens. By wearing shutter glasses, users see the virtual objects in 3D, normally resting on or above the table surface. The user's head position is tracked and the computer-generated stereo image recomputed from that vantage point, so that the virtual objects appear stationary with respect to the physical table, if the user moves their head.

At Fraunhofer IMK, two back projections, one on the table in front of the user and

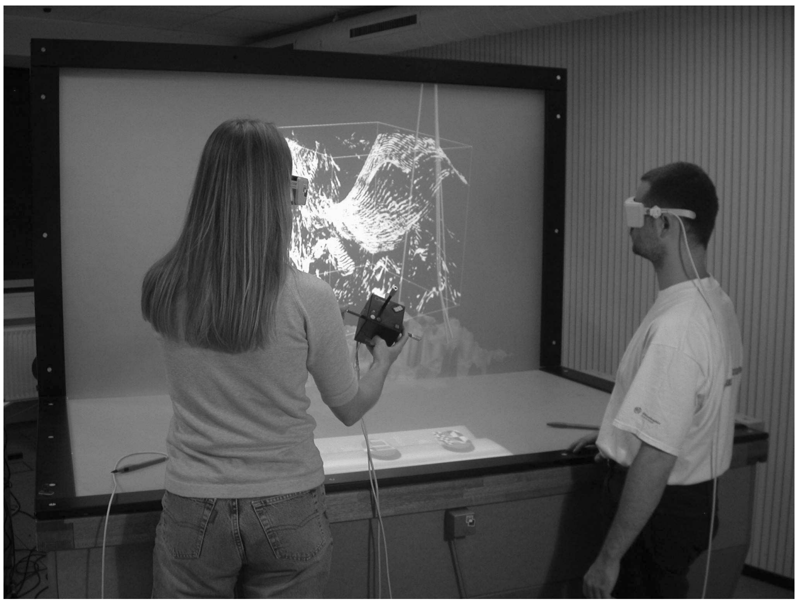


Figure E.1: Responsive Workbench. This image shows two user exploring a model at the Responsive Workbench using the CubicMouse^{TM}.

one on the wall behind it, are used. The interaction space (the space which the user can reach directly, i.e. without moving the model) is between these displays. The workbench was designed for applications which in normal work environments involve work on a desk or on a real workbench, like medical applications, automotive engineering and discussion about architectural models.

The large display surface allows a group of users to interact in this shared workspace face-to-face, which makes local collaboration natural and easy. The visible parts of the virtual objects are mostly within arms length reach of the user, which enables direct manipulation employing both hands. Figure E.1 shows two users exploring a geoseismic model at the Responsive Workbench using the CubicMouse^{TM}, a 6DOF input device.

The RWB is for many applications an appropriate working environment, but it is only semi-immersive, as the user stands on one side of the projection. The Cyber-Stage, described in the next section, is an example of a fully immersive system.

E.2 CyberStage

The CyberStage is Fraunhofer IMK's surround-screen projection-based virtual reality system based on Cruz-Neira's first CAVE installation presented in 1992 [CNSD93]. Stereoscopic images are displayed from the rear onto the three walls and directly onto the floor by Electrohome 8500 projectors. The display resolution is 4x1024x768 pixel. The images are viewed with shutter glasses. The tracking system uses Polhemus Fasttrak sensors.

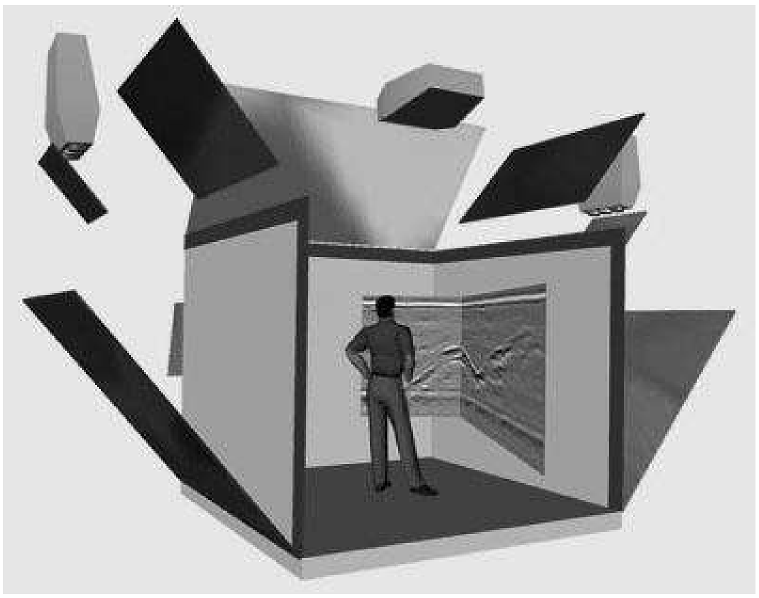


Figure E.2: CyberStage Setup. This image shows the setup of the Fraunhofer IMK Cyber-Stage. Mirrors project the generated stereoscopic images onto the walls.

The tracked user can freely move around and feels immersed in an unbound world. The whole 3m x 3m room represents the interaction space with the virtual world of the application.

The CyberStage allows direct and body centered human interaction as well as team work. Applications are art installations as well as museums, architectural walk-throughs, digital storytelling, and scientific visualization, to name a few. Common interaction tools are the Stylus, a wand like device, and a joystick.

Figure E.2 shows the setup of the CyberStage at IMK with its 4 projectors and mirrors, and Figure E.3 shows a user exploring a museum in the CyberStage.

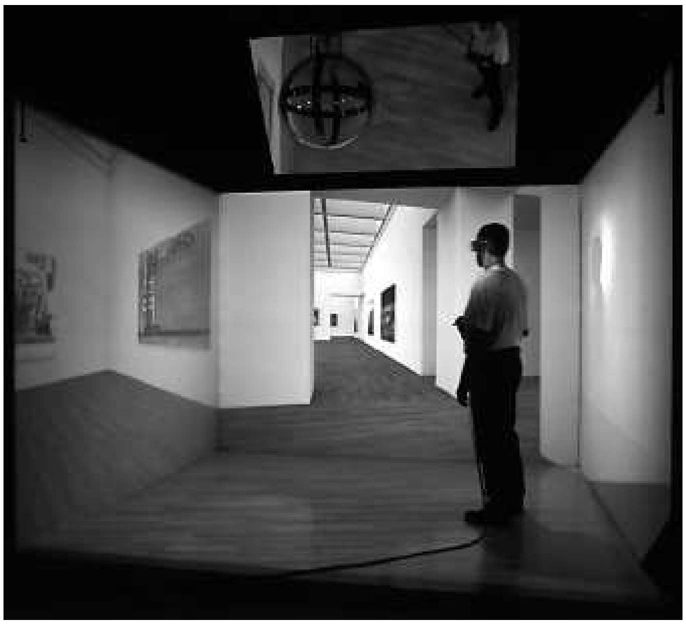


Figure E.3: CyberStage. This image shows a user exploring a museum in the CyberStage. The visible image deformations (discontinuities at the screen boundaries) are due to the physical camera position, as the displayed images are calculated with respect to the position of the user's head (which is motion-tracked). The user navigates the virtual space using a joystick-like control wand.

E.3 i-Cone

The i-Cone (see Figure E.4) is a large cylindric display with a viewing angle of 230 degrees. Four BARCO projectors are used to project the stereo images onto the front of the screen with a resolution of 5500x1320 pixels. The width resolution is less than four times the resolution of the projectors, because the images of the four projectors overlap for blending. Users wear shutter glasses for stereo vision, which are synchronized with infrared emitters. The cone has 9 loudspeakers for sound reproduction. This display allows many users to experience the virtual environment in a wide cinema like environment.

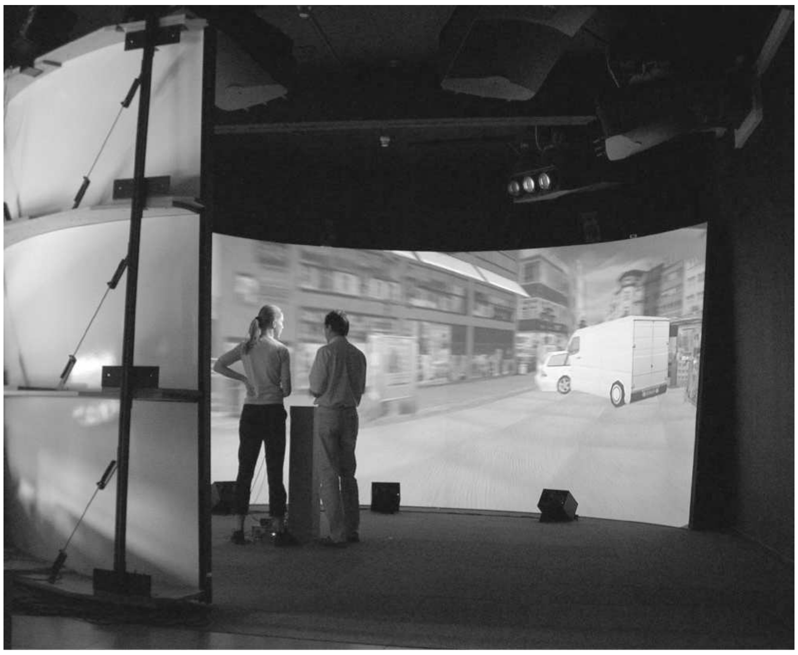


Figure E.4: i-Cone.

The i-Cone is a semi-immersive display. But unlike the RWB and the CyberStage, users are not tracked. The interaction space and the tools used are the same as for the CyberStage.

E.4 AVANGO

AVANGO is Fraunhofer IMK's framework for virtual environment development[Tra99, Tra01]. It is a software which allows fast development of distributed, interactive virtual environment applications for immersive displays like the Responsive Workbench, CAVE, CyberStage and i-Cone.

AVANGO is based on SGI Performer [RH94]. Performer handles advanced rendering tasks like culling, level-of-detail switching, communication with the graphics hardware, and, if available, multi-processing and multi-graphics pipelines. The AVANGO *Nodes* form the object-oriented scene-graph as in Performer. *Sensors* form the interface to devices. Both are programmed in C++. AVANGO also features a complete language binding to the interpreted language Scheme. With Scheme, all AVANGO objects can be created and manipulated on the command line at run-time.

AVANGO includes the following concepts:

Viewer A viewer is the interface between user and virtual environment. It comprises all input and output devices like visual, auditory and tactile displays and spatial trackers.

Scripting Scheme, an interpretive scripting language, is used to change scenes content, viewer features and object behavior at runtime.

Interaction The viewer provides event-based input/output services which can be mapped to objects in the scene.

Extensions With sub-classing the existing C++ system classes, theAVANGO system can easily be extended with new features. This technique will be used later to interface the CubicalPath system with the AVANGO scene-graph.

Streaming All objects can write and read their system to and from a stream. This is the basic facility needed to implement object persistence and distribution

Distribution All objects are distributable and their state is shared by an number of viewers. This allows for collaborative multi-system environments.

F Usability Study

The questionnaire for the usability study in Chapter 8 is presented in F.1. The results of this study are listed in F.2 in the way and language given.

F.1 Questionnaire

Questionnaire I
Virtual Museum – Guided Exploration

General Information:

1. Name:

2. Age:

3. in which professional area do you work (e.g. short job description):

4. Sex:

O female O male

5. Are you:

O right-handed
O left-handed
O ambidextrous (no hand-preference)

6. What kind of learning type are you?

O visual O auditory O verbal O kinestatic O haptic O not known

7. Do you have difficulties to perceive colors?

O no O yes

8. Do you generally get easily motion sick

O Never O sometimes O often

9. If you have ever used Virtual Reality, did you ever get motion sick?

O Never O sometimes O often

10. With which, if any, input devices are you familiar?

O none
O mouse
O joystick
O stylus (Handheld like PalmPilot)
O 3D input devices (like trackers, 3D mice), namely ________________________
O Other: ____

11. Did you ever use Virtual Reality/ experience a Virtual Environment?

O never
O a couple of times
O often

Please describe the kind(s) of Virtual Environment(s) you experienced, if any:

12. If you ever used Virtual Reality, which kind of system did you use?

O Desktop system (with a mouse or joystick and standard monitor)
O Desktop system with a stereoscopic viewing, possibly with 3D input device
O Stereoscopic projection screen (standard or curved)
O Responsive Workbench or similar table-based stereoscopic display system
O CAVE™ or similar room-like stereoscopic multi-wall display system
O Head-Mounted Display
O Augmented Reality (see-through glasses)

Question after self-exploration of the virtual museum

13. Please describe what you remember about the pictures seen
Can you give names, colors and features of pictures?

Questionnaire II

Questions after usage of the supportive system

Name:

**14. Please describe what you remember about the pictures seen
Can you give names, colors and features of pictures?**

15. What do you think about the movement of the camera

a. changes of cameraposition

To fast	just right	to slow

calm	just right	hectic

b. rotation of camera

To fast	just right	to slow

16. What would be your answer, if the camera was interpolated?

To fast	just right	to slow

calm	just right	hectic

17. Did you ever feel dizzy when the supportive system was moving you through the environment

Never	sometimes	always

18. What do you think about the time spent at each picture?

To short just right to long

19. How do you judge the usage of the menu

Easy hard

What is good,

What could be improved

20. Did you ever feel that you did not have the control of the system

Never sometimes always

21. Did you experience the system doing something you didn't <u>intend</u> it to do

Never sometimes always

if so: what was it?

22. Did you experience the system doing something you didn't <u>expect</u> it to do

Never sometimes always

if so: what was it?

23. Did you find the possibility to directly select and move to a picture useful (e.g for a revisit)

Never useful sometimes always useful

24. Did you find the filtering possibility (selecting attributes and period) useful

Never useful sometimes always useful

|—|—|—|—|—|—|

25. Did you feel that you could start and stop the animation in an easy way

Easy hard

|—|—|—|—|—|—|

26. Did you ever feel disoriented after using the supportive system

Never sometimes always

|—|—|—|—|—|—|

27. Did you have the feeling the supportive system was useful to you

a. in contrast to only navigating yourself

Very useful not useful at all

|—|—|—|—|—|—|

b. in giving to you some ideas of what to look at

Very useful not useful at all

|—|—|—|—|—|—|

28. Did you have the feeling the supportive system made you gather more information and gave you a better understanding of the artists work

Yes at some points no

|—|—|—|—|—|—|

29. What did you know about the Philip Gustons work before using the virtual museum?

Nothing a little a lot

|—|—|—|—|—|—|

F.2 Results

User	1	2	3	4	5	6	7	8	9	10
Question										
5	r	r	r	l	r	r	r	r	r	r
6	verbal	visual	visual	?	?	visual	visual	?	visual	visual
7	no	no	no	no	no	no	no	no	no	no
8	no	some	some	no	no	some	often	some	no	no
9	no	no	some	no	no	some	often	some	no	some
10	all	all	all	mouse/joystic	mouse/stylus	all	all	all	mouse/joystic	mouse/joys
11	often	often	often	some	never	often	often	often	some	some
12	all	all	all	desktop	nothing	all	all	all -HMD	Cave/RWB	all
15a1	To fast	fast/right	just right	right/slow	right/slow	right/slow	fast/right	just right	fast/right	fast/right
15a2	hectic	just right	right/hectic	calm/right	just right	right/hectic	hectic	hectic	right/hectic	right/hectic
15b	fast/right	fast/right	just right	right/slow	right/slow	just right	fast/right	right/slow	just right	just right
16a1	fast/right	fast/right	just right	just right	right/slow	right/slow	To fast	To fast	fast/right	fast/right
16a2	calm/right	just right	just right	just right	just right	just right	calm	calm	right/hectic	right/hectic
17	seldom	Never	often	seldom	Never	seldom	often	often	Never	Never
18	To short	short/right	To short	right/long	just right	just right	To short	short/right	To short	short/right
19	ok/hard	Easy	ok/hard	ok/hard	easy/ok	ok/hard	hard	easy/ok	easy/ok	ok/hard
20	sometimes	Never	sometimes	sometimes	Never	Never	never/some	Never	never/some	some/always
21	never/some	never/some	Never	sometimes	Never	Never	Never	Never	never/some	never/some
22	Never	Never	sometimes	never/some	Never	Never	sometimes	Never	Never	never/some
23	never/some	always usefu	Never useful	never/some	sometimes	some/always	never/some	always usefu	always usefu	never/some
24	some/always	Never useful	some/always	never/some	some/always	some/always	always usefu	always usefu	sometimes	Never useful
25	Easy	Easy	easy/ok	ok	Easy	Easy	ok	easy/ok	Easy	easy/ok
26	Never	Never	Never	sometimes	Never	Never	some/always	sometimes	Never	Never
27a	useful	Very useful	useful	useful	useful	Very useful	useful	Very useful	useful	indifferent
27b	Very useful	Very useful	useful	useful	Very useful	Very useful	useful	Very useful	useful	not useful at a
28	Yes	Yes	at some poin	at some poin	mostly yes	mostly yes	mostly no	Yes	mostly yes	mostly no
29	Nothing	a little	Nothing	Nothing	Nothing	Nothing	Nothing	Nothing	Nothing	Nothing
13	all guston; 1949-1972 (in order); 1949-59 (abstract, rote Farbe vornehmlich, Knäuel ais Stricken); 1960s (gegenständ licher); 1970s (wieder abstraktere Darstellunge n	Phillip Guston, some colourful, some more monochrom, abstract, oil?		the curtain - it looked like a cup, dark colors; spleen - it looked like a head, reds oranges; one pictures had red colours and orange	all images of paintings of artists, sometimes bright colors, sometimes dark, remember a cup distinctively, splee, clock, paw		shoe, paw, curtain, flame, untitled period 1948-1979 (he has a nearly photographi cal memory...)	spleen - orange-dominat, a face from one side; paw - a hand hold a pen, yellow; white painting; some figures painted on white backbound	all pictures are of same artist. Chronologic al order (clockwise), nice to see, how the pint techniques evolve during the time; the colors are mostly dark (Phillip Guston)	some kind of impressionis m; colourful; frensh artist
14	The images I' ve seen where mostly from the 1970s Mostly red and black, abstract, with some contenet (shoes, cigars, hats)	grouped by colour or thematic, one picture with cigar, two with shoes, abstract group	Rückkehr, Zone, abstract rot; Gruppe2 abstract rot	the shoe, Phillip Guston, dark round object, 1962; there was a picture of a building, orange/red (PG) ; there was a white picture	lots of dark pictures and a few light ones, very detailed but still easily visible		different categories: massed paint, grid strucures, shoes, egalitaerian structure…	see some abstract pictures and other figurative pictures; sleeping, cigar	read, black, blue, figuraized pictures, (intensions)	hooded men ; arm with cigar; red and dark (abstract?) ; pile of stones and everything (what the system said)
19 (text)	g: looks nice and fancy, the transparenc y is nice! Even that it allows … through the menü I: no … of the fonts in order ro change selection. It gave no feedback wether the selection worked or an image is not there	g: its there I: too big, in the way, sound	I: audio feedback, too hard to actually hit an item, overview of possible choices	g: it is interesting I: the selection tool and the movement (not just forward and backward but side to side		because lack of interpolation	I: menu in front of the images	attribute-classified I: provide a fast way for displaying the label of a picture when stopping the exploration (his problem was that the cppath menu was open but he needed artwork)	filter setting (subjective) I: the interface, much time is spent adjusting values	I: menu structure, option visibility, execute button

User	1	2	3	4	5	6	7	8	9	10
21(text)	sometimes I expected it to move somewhere	one jump when stopping and then		selecting the wrong object with the stylus		strange movement in corner			it did nothing	I pushed the wrong button
23(text)	not really as on can see the pictures in a miniturized form already on the menü					g: many choices and features; easy to read I: difficult to know where to look and click. It needs a better layout				better: directly point to a picture and being moved to it by the system automatically
24(text)										because exhibition experience for me is more general than to view a special attribute
observers comments	extensive use of menu in first pafrgt, guter navigator - benutzt hauptsächlich das supportsystem - neugierde wie es funktioniert	wie gernot	navigation nicht ganz so einfach wie bei frank und gernot - wünscht sich erläuterungen zum Menü	Problems with menu: hitting names and boxes, coordination of buttons on joystick difficult, uses navgation (easier than menu)	good in navigation, well hitting the menu, is always in animation mode, more testing the interface and if knowledgebase gives right results - uses only 1 attribute at a time, goes in navigation mode to be faster, changes often	very good in navigation, looks at pictures one after the other with artworks menu, finished after 3 min - I did not look in gthe second round… bad)	fährt im Kreis herum an den Bildern vorbei - benutzt auch andere Walls (menu vorne, Bildern links) erste durchgänge nur schauen, 3. DG mit Menu - visuelle Qualität zu schlecht für Bilder, Features nicht erkennbar	kommt mir start/stop nicht klar. Navigation schwierig, bräuchte längere Trainingsphase - hatte nichts von dern Bildern selber weil er Probleme mit Steuerung und Menu hatte	stoppt und navigiert selber, restarts - wählt gerne attribute - Probleme mit dem Menu, auswahl schwierig, klickt das menu nicht weg	Steuerung seltsam, läuft selber, sehr gewöhnungsbedürftig, stellt sich selten gerade vor die Bilder - geht in Navigation mode für die Geschwindigkeit. Stopt vor Bildern aber nicht immer - geht dann aber wieder zurück zu den Bildern

other com Gernot: verweildauer an Objekten müßte größer sein; zu 27a the fact that don't have to navigate myself makes it possible to observer the pictures already while I am approaching there.

comments This makes myself more relaxed and free- zu 27b you get more ideas about hidden features one wouldnt recognize without support ; zu 28 because I am not an expert in this field

G Glossary

This glossary lists some of the terms that were used throughout this thesis for clarification and overview.

automated travel Automated travel is automatic motion from the current position to a single target.

camera The camera has a position and an orientation in the scene and specifies the view of the scene. For the CubicalPath system the camera is the representation of the user in the scene.

cube A cube is a volume element of the cube space which holds information as to which objects are situated within this cube.

cube space The cube space is a regular three-dimensional grid of cubes. It represents the geometric scene in a volume data structure.

exploration Exploration is the goal-based motion through the environment for the purpose of discovery, or learning

goal field A goal field is a field consisting of spatial targets with a qualification of their relevancy to a specified goal. In the dynamic potential field method this field maps to the characteristic of the compound attractive potential field.

guided exploration Guided exploration is the independent acquisition of knowledge by exploring the available information. Guidance is provided as a reaction to a user initiated question in form of hints or results of the queries. In virtual environments, these results are presented by moving the user to the spatially distributed geometric representation of the results, one-by-one.

information space The information space is typically a database with a query interface which holds geometric and symbolic information about the objects in the environment. It may also include a semantic network to be able to provide contextual information to each object, thus other objects relevant in the context of this object.

interaction device The hardware part on an interaction tool is the interaction device.

interaction mapping The software part of an interaction tool, which does the translation of hardware sensor values to virtual world behavior.

interaction tasks The purpose for an interaction

interaction tools The combination of an interaction device and mapping.

motion Motion in virtual environments is the act of continuously moving through the environment.

navigation Navigation is the process of moving through an environment.

presentation Automatic motion of the camera through the environment to targets which, by this, are presented to the user.

potential fields Potential field theory has its origin in theoretical physics and describes the behavior of particles in electrostatic fields and the Newtonian attraction between masses. The potential field is a scalar field.

supported exploration The term supported exploration is used synonymously to guided exploration.

travel Travel is the process of moving from a current position to a target location.

user A user in the CubicalPath system is represented by a camera.

view The view specifies, what is seen by the user, thus, what appears in the final image (see Appendix D).

volume data A volume data set consists of S samples $x, y, z, v)$ and is typically arranged in a regular grid.

Curriculum Vitae

Name	Steffi Beckhaus
Address	Vöcklinghauserstr. 16-18 45130 Essen
E-Mail	steffi.beckhaus@imk.fraunhofer.de
Date of Birth	14th of December 1967 in Essen
Nationality	german

Education

1974 - 1978	Albert - Schweitzer Grundschule, Essen
1978 - 1987	Helmholtz - Gymnasium, Essen
1987 - 1991	Universität - GH Essen, Physik DI
1991 - 1992	Gerhard-Mercator-Universität - GH Duisburg, Zusatzstudiengang "Technisches Englisch und Französisch"
1992 - 1993	University of Newcastle upon Tyne, England, Master of Science (M.Sc) in Computing Science
1999 - 2002	Otto-von-Guericke Universität, Magdeburg, registered as p.h.d. student

Qualifications

Jun. 1987	Abitur
Sep. 1991	Diplom Physikingenieurin
Sep. 1993	M.Sc. Computing Science

Work Experience

Oct. 89 - Sep. 92	S. A. S. Systems, Essen, during the studies as an intern, and after graduation as physics engineer
Nov. 93 - Dec. 93	Management Information Services, University of Newcastle, as programmer.
Mar. 94 - Jun. 97	Ing. Büro Messing, Essen, as IT engineer
Mar. 97 - Dec. 98	GMD-Forschungszentrum Informationstechnik, IMK.Delta, as p.h.d student
Jan. 99 - Dec. 01	GMD-Forschungszentrum Informationstechnik, IMK.VE, as p.h.d. student
since Jan. 02	Fraunhofer IMK.VE (GMD-FhG fusion), as a research scientist

Publications

S. Beckhaus, J. Wind, and T. Strothotte
Hardware-Based Voxelization for 3D Spatial Analysis.
In Proceedings of CGIM '02. (to appear), Kauai, Hawai, August 2002.

S. Beckhaus, F. Ritter, and T. Strothotte
Guided exploration with dynamic potential fields: The CubicalPath System.
In Computer Graphics Forum, volume 20(4), pages 201–210. The Eurographics Association 2001, Blackwell Publishers, December 2001.

S. Beckhaus, G. Eckel, and T. Strothotte
Guided exploration in virtual environments.
In Woods, A. J., Bolas, M. T., Merrit, J. O., and Benton, S. A., editors, Stereoscopic Displays and Virtual Reality Systems VIII, Proceedings of Electronic Imaging '01, volume 4297, pages 426–435. San José, CA, SPIE Press, Bellingham, WA, July 2001.

G. Eckel, and S. Beckhaus
Exviz: A virtual exhibition design environment.
In Proceeding of the International Symposium on Virtual and Augmented Architecture (VAA '01), pages 171–182, Dublin, Ireland, Springer Verlag, June 2001.

S. Beckhaus, F. Ritter, and T. Strothotte
Cubicalpath – dynamic potential fields for guided exploration in virtual environments.
In Proceedings of Pacific Graphics '00, Hong Kong, China, pages 387–395, IEEE Society Press, Los Alamitos, California, November 2000.

Presentations

S. Beckhaus, J. Wind
Non-Graphical Application of Hardware-Accelerated Voxelization.
In Sketches & Applications of SIGGRAPH '01, page 237 , Los Angeles, California, August 2001.

Bibliography

[AF00] Alonso, M. and Finn, E. J. *Physik.* Oldenbourg, München, 3rd. edition, 2000.

[Ari76] Arijon, D. *Grammar of the Film Language.* Silman-James Press, 1976.

[BD95] Bortz, J. and Döring, N. *Forschungsmethoden und Evaluation.* Springer-Verlag, 1995.

[BDHB98] Bowman, D. A., Davis, E. T., Hodges, L. F., and Badre, A. N. Mantaining spatial orientation during travel in an immersive virtual environment. *Presence*, 7(3):225–240, 1998.

[BH99] Bowman, D. A. and Hodges, L. Formalizing the design, evaluation, and application of interaction techniques for immersive virtual environments. In *Journal of Visual Languages and Computing*, pages 37–53. Academic Press, October 1999.

[BKH97] Bowman, D. A., Koller, D., and Hodges, L. F. Travel in immersive virtual environments: An evaluation of viewpoint motion control techniques. In *Proceedings of the Virtual Reality Annual International Symposium (VRAIS)*, pages 45–52, 1997.

[BL89] Barraquand, J. and Latombe, J. C. Robot motion planning: A distributed representation approach. Technical report, Department of Computer Science, Stanford University, 1989.

[BLP83] Brooks, R. A. and Lozano-Peréz, T. A subdivision algorithm in configuration space for findpath with rotation. In *Proceedings of the 8th International Conference on Artificial Intelligence*, pages 799–806, Karlsruhe, 1983.

[Bow02] Bowman, D. A. *Handbook of Virtual Environments : design, implementation, and applications*, chapter 13, pages 239–254. Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey 07430, 2002.

[Bro83] Brooks, R. A. Solving the find-path problem by good representation of free space. *IEEE Transactions on Systems, Man and Cybernetics*, 13(3):190–197, 1983.

[Bro88] Brooks, F. P. Grasping reality through illusion – interactive graphics serving science. In *Proceedings of CHI'88*, pages 1–11, New York, May 1988.

[BRS01] Beckhaus, S., Ritter, F., and Strothotte, T. Guided exploration with dynamic potential fields: The CubicalPath System. In *Computer Graphics Forum*, volume 20(4), pages 201–210. The Eurographics Association 2001, Blackwell Publishers, December 2001.

[Bru76] Bruner, J. S. *The Process of Education.* Harvard University Press, 1976.

[BS91] Bronstein, I. N. and Semendjajew, K. A. *Taschenbuch der Mathematik.* Ed. G. Grosche and V. Ziegler and D. Ziegler, B. G. Teubner Verlagsgesellschaft, Stuttgart, Leipzig, 1991.

[BS96] Billinghurst, M. and Savage, J. Adding intelligence to the interface. In *Proceedings of the 1996 Virtual Reality Annual International Symposium (VRAIS '96).* IEEE, 1996.

[BT98] Bandi, S. and Thalmann, D. Space discretization for efficient human navigation. In *Proceedings of Eurographics*, volume 17, pages 195–206, March 1998.

[BW90] Baum, D. R. and Winget, J. M. Real time radiosity through parallel processing and hardware acceleration. *Computer Graphics (Symposium on Interactive 3D Techniques)*, 24(2):67 – 75, March 1990.

[BW95] Baker, M. P. and Wickens, C. D. Human factors in virtual environments for the visual analysis of scientific data. Technical report, NCSA-TR032, Institute of Aviation, August 1995.

[BWS02] Beckhaus, S., Wind, J., and Strothotte, T. Hardware-Based Voxelization for 3D Spatial Analysis. In *Proceedings of CGIM '02.* (to appear), August 2002.

[Can88] Canny, J. F. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA., 1988.

[CKY00] Chen, M., Kaufmann, A. E., and Yagel, R. *Volume Graphics.* Springer Verlag, 2000.

[CNSD93] Cruz-Neira, C., Sandin, D. J., and DeFanti, T. A. Surround-screen projection-based virtual reality: The design and implementation of the cave. *Proceedings of SIGGRAPH 93*, pages 135–142, August 1993.

[Coh94] Cohen, D. Voxel traversal along a 3D line. In Heckbert, P. S., editor, *Graphics Gems IV*, pages 366–369. Academic Press, Inc., 1994.

[Con01] Conroy, R. *Virtual Navigation in Immersive Virtual Environments.* PhD thesis, London University College, 2001.

[CW93] Cohen, M. F. and Wallace, J. R. *Radiosity and Realistic Image Synthesis.* Academic Press Professional, Boston, MA, 1993.

[DC97] Darken, R. P. and Cockayne, W. R. The omni-directional treadmill: a locomotion device for virtual worlds. In *Proceedings of UIST*, pages 213–221, 1997.

[DM94] Durlach, N. I. and Mavor, A. S. *Virtual Reality: scientific and technological challenges*. National Academy press, 1994.

[DP02] Darken, R. P. and Peterson, B. *Handbook of Virtual Environments: design, implementation, and applications;*, chapter 24, pages 239–254. Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey 07430, 2002.

[Dru94] Drucker, S. M. *Intelligent Camera Control for Graphical Environments*. PhD thesis, Massachussetts Institute of Technology, July 1994.

[DZ94] Drucker, S. M. and Zeltzer, D. Intelligent camera control in a virtual environment. In *Proceedings of Graphics Interface*, pages 190–199, Banff, Alberta, Canada, 1994. Canadian Information Processing Society.

[EB01] Eckel, G. and Beckhaus, S. Exviz: A virtual exhibition design environment. In *Proceeding of the International Symposium on Virtual and Augmented Architecture (VAA '01)*, pages 171–182, Dublin, Springer Verlag, June 2001.

[Ede94] Edelstein, H. Unraveling client/server architectures. *DBMS*, 7(5):34, May 1994.

[FC00] Fang, S. and Chen, H. Hardware accelerated voxelization. *Computers and Graphics*, 24(3):433–442, June 2000.

[Fur86] Furnas, G. W. Generalized fisheye views. In *Proceedings of CHI '86, Human Factors in Computing Systems*, pages 16–23. ACM SIGCHI, 1986.

[FvDFH90] Foley, J. D., van Dam, A., Feiner, S. F., and Hughes, J. F. *Computer Graphics: Principles and Practise*. Addison-Wesley Systems Programming Series. Addison-Wesley Publishing Company, 2nd edition, 1990.

[Gib98] Gibson, S. F. F. Using distance maps for accurate surface representation in sampled volumes. In *IEEE Symposium on Volume Visualization*, pages 23–30, 1998.

[Han97] Hand, C. A survey of 3D interaction techniqies. *Computer Graphics Forum*, 16(3):269–281, 1997.

[Har01] Hartmann, K. *Text-Bild-Beziehungen in multimedialen Dokumenten: Eine Analyse aus Sicht von Wissensrepräsentation, Textstruktur und Visulasierung*. PhD thesis, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, December 2001.

[HCS96] He, L., Cohen, M. F., and Salesin, D. H. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. *Proceedings of SIGGRAPH 96*, pages 217–224, August 1996.

[HH93] Hix, D. and Hartson, H. R. *User Interface Development: Ensuring Usability through Product and Process.* New York: John Wiley and Sons, 1993.

[HMK$^+$97] Hong, L., Muraki, S., Kaufman, A. E., Bartz, D., and He, T. Virtual voyage: Interactive navigation in the human colon. *Proceedings of SIGGRAPH '97*, pages 27–34, August 1997.

[Hol02] Hollerbach, J. M. *Handbook of Virtual Environments : design, implementation, and applications*, chapter 11, pages 239–254. Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey 07430, 2002.

[HV99] Henning, M. and Vinoski, S. *Advance CORBA Programming with C++.* Addison-Wesley professional computing series. Addison Wesley Longman, Inc., Reading, Massachusetts, 1999.

[HW97] Hanson, A. J. and Wernert, E. A. Constrained 3D navigation with 2d controllers. In *Proceedings of Visualization '97*, pages 175–182. IEEE Computer Society Press, 1997.

[Jon96] Jones, M. W. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, 15(5):311–318, 1996.

[Kat86] Kathib, O. Real-time obstacle avoidance for manipulators and mobile robots. In *International Journal of Robotics Research*, volume 5(1), pages 90–99, 1986.

[Kat91] Katz, S. D. *Film directing shot by shot: visualizing from concept to screen.* Michael Wiese Productions, 1991.

[Kau87a] Kaufman, A. An algorithm for 3D scan-conversion of polygons. In *Proceedings of Eurographics*, pages 197–208, August 1987.

[Kau87b] Kaufman, A. Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes. *Computer Graphics (Proceedings of SIGGRAPH '87)*, 21(4):171–179, July 1987.

[KF90] Karp, P. and Feiner, S. Issues in the automated generation of animated presentations. *Proceedings of Graphics Interface*, pages 39–48, 1990.

[KF93] Karp, P. and Feiner, S. Automated presentation planning of animation using task decomposition with heuristic reasoning. *Proceedings of Graphics Interface*, pages 118–127, 1993.

[KF94] Krüger, W. and Fröhlich, B. The responsive workbench. *IEEE Computer Graphics & Applications*, 14(3):12–15, May 1994.

[KMH95] Kolb, C., Mitchell, D., and Hanrahan, P. A realistic camera model for computer graphics. *Proccedings of Siggraph '95*, pages 317–324, 1995.

[Lat91] Latombe, J. C. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[Lav95] Lavalle, S. M. *A Game-Theoretic Framework for Robot Motion Planning.* PhD thesis, University of Illinois at Urbana-Champaign, Illinois, 1995.

[LP81] Lozano-Peréz, T. Automatic planning of manipulator transfer movements. *IEEE Transactions on Systems, Man and Cybernetics*, 11(10):681–698, 1981.

[LRDG90] Lengyel, J., Reichert, M., Donald, B. R., and Greenberg, D. P. Real-time robot motion planning using rasterizing computer graphics hardware. *Computer Graphics*, 24(4):327–335, 1990.

[MCR90] Mackinlay, J. D., Card, S. K., and Robertson, G. G. Rapid controlled movement through a virtual 3D workspace. *Computer Graphics (Proceedings of SIGGRAPH '90)*, 24(4):171–176, August 1990.

[Mer] Merriam-Webster. Merriam-Webster Collegiate Dictionary. http://www.m-w.com/.

[Min95] Mine, M. R. Virtual environment interaction techniques. Technical Report 95-018, Department of Computing Science, University of North Carolina, Chapel Hill, NC 27599-3175, 1995.

[Nie93] Nielson, J. *Usability Engineering.* Academic Press, 1993.

[Nil69] Nilsson, N. J. A Mobile Automation: An Application of Artificial Intelligence Techniques. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pages 509–520, Washington D.C., 1969.

[OEW] OEW. Innovative software GmbH. http://www.isg.de.

[OMG] OMG. CORBA specification. http://www.omg.org.

[ÓSY83] Ó'Dúnlaing, C., Sharir, M., and Yap, C. K. Retraction: A new approach to motion planning. In *Proceedings of the 15th ACM Symposium on the Theory of Computing*, pages 207–220, Boston, 1983.

[Pap65] Papouls, A. *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill, 1965.

[Pre98] Preim, B. *Interaktive Illustration und Animation zur Erklärung komplexer räumlicher Zusammenhänge.* PhD thesis, Otto-von-Guericke Universität, Magdeburg, 1998.

[PST$^+$96] Pausch, R., Snoddy, J., Taylor, R., Watson, S., and Haseltine, E. Disney's aladdin: First steps towards storytelling in virtual reality. *Proceedings of Siggraph '96*, pages 193–203, 1996.

[Rat] Rational Rose. http://www.rational.de.

[RBD$^+$99] Ritter, A., Böttger, J., Deussen, O., König, M., and Strothotte, T. Hardware-based rendering of full-parallax synthetic holograms. *Applied Optics*, pages 1364–1369, 1999.

[RH94] Rohlf, J. and Helmann, J. Iris Performer: A high performance multiprocessing toolkit for real-time 3D graphics. In *Proceedings of SIGGRAPH '94*, pages 381 – 394. ACM Press New York, NY, USA, 1994.

[Rhe91] Rheingold, H. *Virtual Reality.* Secker & Warburg, 1991.

[RJ97] Rickel, J. and Johnson, W. L. Intelligent tutoring in virtual reality: A preliminary report. In *Proceedings of the 8th World Conference on AI in Education*, August 1997.

[RPDS00] Ritter, F., Preim, B., Deussen, O., and Strothotte, T. Using a 3d puzzle as a metaphor for learning spatial relations. In *Proceedings of Graphics Interface*, pages 171–178, Montréal, Que., Canada, 2000.

[RS97] Regenbrecht, H. T. and Schubert, T. W. Measuring presence in virtual environments. In *HCI International*, July 1997.

[Rub94] Rubin, J. *Handbook on usability testing: how to plan, design, and conduct effective tests.* Wiley technical communication library, 1994.

[SA] Segal, M. and Akeley, K. OpenGL Specifications and State Machine Diagram. http://www.opengl.org/developers/documentation/specs.html.

[SCP95] Stoakley, R., Conway, M. J., and Pausch, P. Virtual reality on a WIM: Interactive worlds in miniature. In *Proceedings of CHI'95*, pages 265–272. ACM SIGCHI, March 1995.

[Sho85] Shoemaker, K. Animating rotation with quaternion curves. In *Proccedings of SIGGRAPH '85*, volume 19(3), July 1985.

[SK99] Sramek, M. and Kaufman, A. E. Alias-free voxelization of geometric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):251–267, July 1999.

[Sla99] Slater, M. Measuring presence: A response to the Witmer and Singer presence questionnaire. *Presence*, 8(5):560–565, 1999.

[SSH87] Schwartz, J. T., Sharir, M., and Hopcroft, J. *Planning, Geometry, and Complexity of Robot Motion.* Ablex, Norwood, NJ., 1987.

[Sta02] Stanney, K. M. *Handbook of Virtual Environments : design, implementation, and applications.* Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey 07430, 2002.

[Str98] Strothotte, T. *Computational Visualization: Graphics, Abstraction, and Interactivity.* Springer-Verlag Heidelberg, Berlin, Heidelberg, 1998.

[Sut63] Sutherland, I. E. Sketchpad - a man-machine graphical communication system. Technical report, No. 296, MIT Lincoln Laboratory, 1963.

[SW97] Slater, M. and Wilbur, S. A framework for immersive virtual environments (five): Speculations on the role of presence in virtual environments. *Presence: Teleoperators and Virtual Environments*, 6(6):603–616, 1997.

[Swe88] Sweller, J. Cognitive load during problem solving : Effects on learning. *Cognitive Science*, 12:257–285, 1988.

[Swe94] Sweller, J. Cognitive load theory, learning difficulty and instructional design. *Learning and Instruction*, 4:295–312, 1994.

[TBGT91] Turner, R., Balaguer, F., Gobeletti, E., and Thalmann, D. Physically-based interactive camera motion control using 3D input devices. In *Proceedings of Computer Graphics International '91*, pages 135–145, 1991.

[TBN00] Tomlinson, B., Blumberg, B., and Nain, D. Expressive autonomous cinematography for interactive virtual environments. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 317–324, Barcelona, Catalonia, Spain, 2000.

[Tra99] Tramberend, H. Avocado: A distributed virtual reality framework. In Rosenblum, L., Astheimer, P., and Teichmann, D., editors, *Proceedings of IEEE Virtual Reality*, pages 14–21, Los Alamitos, California, 1999.

[Tra01] Tramberend, H. A display device abstraction for virtual reality applications. In Chalmers, A. and Lalioti, V., editors, *Proceedings of AFRIGRAPH '01*, Capetown, South Africa, November 2001.

[WE98] Westermann, R. and Ertl, T. Efficiently using graphics hardware in volume rendering applications. *Proceedings of SIGGRAPH '98*, pages 169–178, July 1998.

[WH99] Wernert, E. A. and Hanson, A. J. A framework for assisted exploration with collaboration. In Ebert, D., Gross, M., and Hamann, B., editors, *IEEE Visualization '99*, pages 241–248, 1999.

[WK93] Wang, S. and Kaufman, A. Volume sampled voxelization of geometric primitives. In *Proceedings of Visualization'93*, pages 78–84. IEEE, 1993.

[WNDO99] Woo, M., Neider, J., Davis, T., and Open Architecture Review Board. *OpenGL Programming Guide.* Addison-Wesley Publishing Company, 3rd. edition, 1999.

[WO90] Ware, C. and Osborne, S. Exploration and virtual camera control in virtual three dimensional environments. In *Proceedings of 1990 Symposium of Interactive 3D Graphics*, pages 175–183. ACM Siggraph, 1990.

[WS94] Witmer., B. and Singer, M. Measuring immersion in virtual environments. Technical Report 1014, U.S. Army Research Institute for the Behavioral and Social Sciences, Alexandria, VA, 1994.

List of Figures

List of Tables