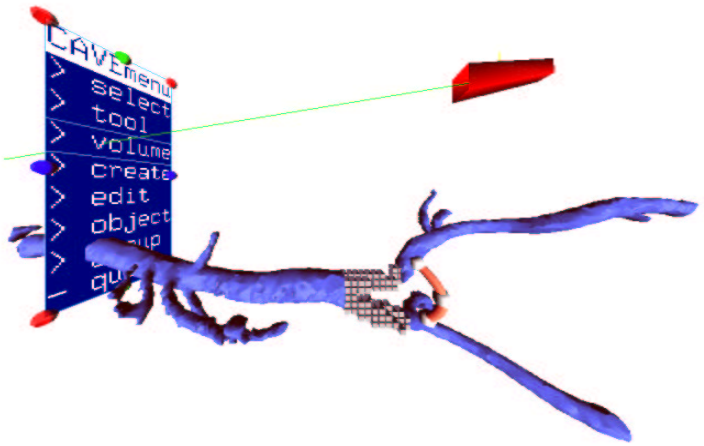
Interaction in Virtual Reality



Don Hannema

Begeleiders: drs. Robert G. Belleman
prof. dr. Peter M.A. Sloot

Keywords: immersive virtual reality, context sensitive speech recognition,
multi-modal interaction toolkit

Interaction in Virtual Reality

Doctoraal Informatica,
Specialisatie Technische Informatica,
Faculteit der Natuurwetenschappen, Wiskunde en Informatica,
Sectie Computational Science,
Universiteit van Amsterdam

augustus 2001

Contents

Chapter 1

Introduction

1.1 Overview

Computer simulations are widely used in industrial and scientific applications. Large scale computing techniques, used in e.g. climate modelling, simulations in fluid dynamics and bio-molecular modelling, but also acquisition devices, such as CT (computed tomography) use the steadily increasing computer power to create complex and often time-dependent data.

1.1.1 Scientific visualization and virtual reality

Most of this data is too vast to analyze analytically or numerically. Scientific visualization provides a way to quickly gain insight in the information the data contains. From a computing perspective, scientific visualization is part of a greater field called visualization, which involves other research areas, such as computer graphics, image processing and high performance computing. Scientific visualization provides a vehicle for exploring the data through a visual representation.

Scientific visualization of multi-dimensional data is possible with virtual reality (VR) which is the experience of perceiving a synthetically generated environment as if it is real. With VR, visualization is effectively extended from a flat, two dimensional projection to a full, three dimensional interactive representation, which allows the user to explore a data set more efficiently.

A virtual reality system (i.e. special hard and software) is required to actually achieve VR. Generally, the display of VR takes the form of a projection on a 2D desktop monitor, as is the case in the gaming industry [6]. However, with this kind of VR it is not possible to directly interact with synthesized objects in the generated environment. A virtual environment (VE) allows a person to be surrounded by the 3D computer-generated world and to move around the virtual world and see it from different angles in an intuitive way [9]. This synthetic environment, generated by software running on a VR system, will greatly enhance the "feeling of presence" or "immersion".

1.1.2 Computational steering

The traditional cycle in simulation is to prepare input, execute a simulation, and to visualize the results as a post-processing step. However, more insight may be obtained if these activities are done simultaneously. Computational steering is very useful when one has the possibility to interactively change the simulation parameters while the simulation is in progress and immediately see the effect of this change through the resulting new data [28, 41].

As changes in parameters become more instantaneous, the cause-effect relationships within the simulation become more evident, allowing the scientist to develop more intuition about the effect of problem parameters, to detect program bugs, to develop insight into the operation of an algorithm, or to deepen an understanding of the physics of the problem being studied.

1.2 Interactive Virtual Environments

Visualization of complex structures in VR can be performed with a wide variety of existing visualization applications [14, 26, 34]. The ability to manipulate these structures in virtual environments, however, is still often inadequate and underdeveloped for many reasons. Some of these problems are of a technological nature. For example, improving the accuracy and resolution of tracking devices aids precise positioning and movement of virtual objects. Higher display resolution and more computational power provides the ability to increase the amount of information that can be visualized (providing more detail in the visualized data) and to reduce delays (resulting in a smoother, more responsive interaction experience for the user).

Improving technology alone does not provide sufficient means to create intuitive interaction techniques (ITs). As a lot of research has already been focused on technological problems, there is now more focus at solving the problems that are of a conceptual nature. ITs rely on a mapping between an action in the physical world and the resulting action in the virtual world. Care must be taken that these mappings do not contradict the user's mental representation and expectation of a specific action, or in other words, ITs should be as intuitive as possible. This implies that mappings should not deviate too much from those used in the real world.

1.2.1 Problems with interaction in VEs

One of the major problems in VEs is the lack of physical constraints [23, 24, 32]. Interaction in the real world is aided by physical properties of objects around us. Two of the most important include adherence to gravity and not being able to move through solids. Virtual environments do not have any of such restrictions. Picking up an object is therefore difficult in a VE. Most constraints (e.g. collision detection) can be simulated, but involve the calculation of many intersections.

Moreover, tracking devices are often rather expensive, which forces VEs to only use a few of them. This restricts the amount of possible input from the user and therefore the usability of the system. Also, the limited precision of input devices restricts the accuracy in which manipulations can be done. Frequently, readings from input devices change their value constantly when they should remain steady (this is often the result of "drift" in the acquisition technique that is used).

Regular two dimensional desktop computers use the WIMP (windows, icons, mouse and pointer) metaphor as the standard user interface. This interface is commonly used on a wide variety of platforms, making it possible for users to immediately start working with familiar interface elements. Virtual environments do not have a common user interface, which is mainly because there is no standard set of input devices.

1.3 Goal of this research

This thesis addresses some of the hardware and software requirements required for interaction in a VE. Chapter 2 describes various aspects of virtual environments in detail, including tracking devices, navigation and wayfinding, and various other interaction techniques (ITs) commonly used in VEs.

Speech recognition technology is believed to have matured enough to provide a more natural interface to virtual environments. Context sensitivity in particular is believed to provide the required constraints for reliable recognition. Chapter 3 describes a context sensitive speech recognition system and discusses a user study on its performance in a virtual environment.

The design and implementation of a multi-modal interaction toolkit is presented in chapter 4, which incorporates context sensitive speech recognition as one of the available interaction techniques. To validate the interaction techniques that have been developed, the application of these techniques are validated in two test cases. The first test case focusses on the efficiency with which a CT scan can be edited. The second test case is part of a more ambitious plan to add an interaction component to a virtual vascular reconstruction operating theater, which is under development at the Section Computational Science of the University of Amsterdam. Finally, chapter 5 presents conclusions and points out areas of further research.

1.4 Acknowledgements

The coral datasets in this thesis are used with permission by Mark Vermeij (IBED, University of Amsterdam), Rolf Bak (NIOZ) and Leo Lampmann (St. Elisabeth Hospital, Tilburg). The abdominal artery dataset is courtesy of Charles Taylor from Stanford University.

Chapter 2

Immersive Virtual Exploration Environments

2.1 Introduction

Immersive Virtual Exploration Environments (IVEEs) are environments that immerse a user into a computer generated world and allow the user to interact with the environment. This chapter describes the issues involved in creating a useful IVEE.

The visualization of complex scenes in VR, which consist of very many different or large objects, can be performed with a wide variety of existing visualization applications that enable an explorer to e.g. view virtual objects from different angles [14, 26, 34]. However, such applications merely provide a "walkthrough" and no means to directly interact with, e.g. select and manipulate, virtual objects in the scene. With help of input devices it becomes possible to overcome this problem and be able to interact with virtual objects in the virtual environment (VE). The same input device can be used with various interaction techniques, making it possible to do many different tasks.

Immersive manipulation is similar to manipulation in the real world, but there are some significant differences which have to be understood in order to exploit the full potential of VR technology. For example, users can perform actions which are impossible in the real world (e.g. walk through a virtual wall). Furthermore, VEs have no direct analogy to the familiar desktop interaction metaphor, making it difficult to implement and perform basic interaction tasks, such as selection and manipulation, without alternatives.

Full exploration of a large VE is only possible if the user is able to navigate through the environment. Often the VE is larger than the physical environment or beyond the tracking capabilities of the tracking system [21]. These large VEs require navigation techniques to allow exploration without any physical movement by the user.

2.2 Architecture of VR systems

Fully immersive virtual environments allow one to use hands, eyes and ears in much the same way as in the real world. This way, a human being can be "fooled" into believing that

he or she is located in a computer generated environment. The difference is that all sensory stimuli are computer generated. It is therefore necessary to add special hardware to the VR system such that these stimuli can be directed to the user. Intention of the user is made clear to the computer via (tracked) input devices.

2.2.1 Sensory rendering

Of the human sensory system (i.e. vision, hearing, smell, taste and touch), the primary sense to be rendered in a VE is vision.

Visual rendering

Convincing visual input is generated by tricking the brain into believing it is looking at three dimensional objects. This is made possible due to stereopsis, which is the ability of the brain to fuse two slightly different images (i.e. from two slightly different perspectives, corresponding to the difference in perspective that the left and right eye perceive) into a mental picture of three dimensions.

There are several depth cues that help us in the real world. Among these are occlusion (which provides clues as to which object is farther away) but also e.g. lighting and shadows give helpful depth information. Furthermore, feedback from changes in the scene due the user's head movement (motion parallax) and rotation of the eyes to the center of interest (convergence) are important depth cues [9]. Besides immediate visual feedback when the user moves his head, visual cues like bounding boxes or changing color all help in the sense that actions have been understood by the system.

When the structures to be visualized are not too complex, the computer will be capable of rapidly generating stereoscopic images. To keep eye fatigue to a minimum, images must be sent to each eye at a frequency of at least sixty frames per second. The actual update of the images is allowed to be as low as ten updates per second. Lower update-rates make the movements of objects too jerky for any real practical interaction. Smooth movements are perceived if the update-rate is at least twenty five times per second.

There are various ways to feed the eyes with with computer generated visual impulses. Typically the distinction is made between a head-mounted display (HMD) and immersive projection technology (IPT). In a HMD, the display is attached to a helmet that is worn by the user. It consists of two small color LCD screens mounted in front of each eye. HMD devices are expensive and, although they are getting smaller, also heavy to wear. Moreover, HMDs frequently cause motion sickness.

IPT uses a stationary display, e.g. a monitor or projection display attached to a wall, where only the eye points move. One method to direct stereo pair images into the correct eye is to use projectors equipped with polarized filters, with corresponding filters placed in front of both eyes. The most common way is to use liquid crystal shutter glasses (figure 2.1) which alternately block incident light to either eye in sync with the images generated by the display.

The main difference between these two display methods is the possibility to work in a team of researchers. HMDs have completely opaque displays, which makes it impossible for team members to see each other let alone point at, or discuss objects of interest [22]. The

Figure 2.1: Liquid crystal shutter glasses.

advantage of polarized or shutter glasses is that they are semi-transparent. This will allow team members to see each other and point at interesting (virtual) objects. In projection based systems, each researcher will see each other and the virtual world at the same time, making it easier to discuss findings. In this thesis, we restrict ourselves to VR systems that use IPT for visual rendering.

Auditory rendering

Sound is another commonly simulated sensory input. This "auditory rendering" will often greatly enhance the feeling of immersion. Although mono and stereo sound is sometimes sufficient to provide audible feedback, it can be desirable to know exactly where a particular sound originated from. This spatially localized audio allows one to quickly determine from which particular object the sound emanated from. An example of a spatially localized audio library is OpenAL [35], the Open Audio Library. This is however still difficult, because sound tends to bounce of physical objects (such as walls) resulting in reverberation [9, 39].

Sound is very effective to accentuate the interaction experience. This auditory feedback could be as simple as a "tick" when an object is first touched and a "tock" when the object is not touched [42]. Typically these are sounds of 100 to 500 milliseconds long, but they can also loop. For example, dragging your virtual hand over sand would sound different than dragging it through gravel.

Speech synthesis, i.e. the ability to allow the computer to provide verbal feedback, can provide useful and easy to comprehend feedback. It is often possible to make a selection from a wide variety of languages. The quality of synthesized text is not yet perfect, because it lacks emotion and frequently sounds tinny [46]. Fortunately, this is not a real problem for the human interpreters.

Other senses

There are other senses that can be stimulated, including the sense of balance, motion, and temperature. Some of these senses can be very useful for specific applications. For example, the sense of balance is often exploited using motion platforms in commercial flight simulators. The sense of smell and taste are rarely used in VEs.

Manipulation of objects can become a lot easier when one is able to "feel" texture, shape and size of a virtual object. Tactile and force feedback techniques are required that allow some sense of touch. For example, tactile feedback can be rendered through a glove fitted with small balloons, which quickly fill and deflate to provide touch sensation to the hand.

Other sensory stimuli are still too difficult to render efficiently. For example, the sensation of heat and cold can be simulated in a localized area with very small electrical heat pumps, but these systems are fairly expensive and may require a full body suit equipped with many heat pumps.

2.2.2 Input devices

Input (or control) devices are another necessity for interactive VEs to allow a user to interact with virtual objects. Input devices often consist of a number of buttons and/or joysticks embedded in a hand-held device referred to as a "controller". Actions performed with a controller have a global meaning, making it impossible to define a relation to a specific object. An example of this are buttons, analog joysticks for smooth speed-sensitive movement or scroll-buttons for step-wise manipulation. These discrete manipulators provide a reliable way for the user to show intention to the environment as a whole.

Input devices that use tracking technology to determine position and orientation are referred to as "manipulators". A manipulator that is commonly associated with virtual reality is the data glove. This global input device is made up of a glove fitted with sensors in each finger to measure flex and abduction (the "spread" between fingers). Interaction with the virtual environment is done via gestural commands, which are a direct mapping of the real world to the virtual world. In contrast to simple manipulators like buttons, data gloves allow far more freedom in expression. This, however, also makes them more difficult to use. Although tracked control devices are the most commonly used, every device attached to a computer that sends properties of the real world to the virtual world can function as an input device (as for example a microphone, mouse or keyboard).

Tracking sensors

One key element for interaction with a virtual world is a method to track the position of real world objects. These tracking sensors can establish a relation between objects in the real world and virtual objects so that interaction becomes possible. All physical objects can be tracked, but two are the most common. The first is tracking of the head to determine the direction of sight. This makes it possible to use motion parallax as a depth cue, but will also determine which objects are in the field-of-view (FoV). The other commonly tracked object is one or more limbs.

Because tracking is important, it must be as accurate as possible. Ideally trackers consist of six degrees of freedom (DOF) sensors for position (X, Y and Z coordinates) and orientation (pitch, roll and yaw). Tracking devices vary largely in accuracy and cost. The biggest problem for position tracking are accuracy and latency. In the ideal case, position and orientation are known to the simulating computer the instant the user makes a movement. There are many tracker-based input devices on the market today, all with their own advantages and

Figure 2.2: The CAVE Automated Virtual Environment located at SARA Amsterdam.

disadvantages. For example, the "wand"; a three dimensional mouse containing three buttons and a joystick. Table 2.1 gives an overview of several tracking techniques.

method	*technique*	*accuracy*	*latency*	*remarks*
mechanical	armatures	very high	near zero	restricts user's motion
acoustic	sound	middle	high	sensitive to room acoustics
magnetic	EM field	high	low	sensitive to ferro-materials, rapid decrease in accuracy with distance
optical	optics	high	high	requires line of sight, cost is high
inertial	acceleration	high	very low	no interference problems, but suffers from inaccuracy over longer periods of use, due to second order integration

Table 2.1: Overview and comparison of several tracking techniques.

2.2.3 Example: CAVE

An example of an immersive virtual environment is the CAVE Automated Virtual Environment. One of these VR systems is located at SARA in Amsterdam [33]. It is an IPT system, consisting of a 3m x 3m x 3m room with three rear-projection screens for walls and a down-projection screen for the floor (see figure 2.2).

The core of the system is an SGI Onyx2 Reality Monster with four Infinite Reality graphics pipelines, 8 MIPS R10000 196 MHz processors, 1 GB main memory and a HIPPI network interface of 800 MB/s. Each display has a resolution of 1024x768 pixels per wall at a refresh rate of 60 Hz full color anti-aliased stereo. The main viewer wears a pair of electromagnetically tracked stereo shutter glasses (i.e. Stereographics' CrystalEyes), such that the correct stereoscopic projections are calculated for that wearer, and a tracked wand [9].

2.3 Object interaction

Object interaction is a combination of several basic tasks, similar to the ones used in the real world [11, 16, 21, 29]. These tasks include selection, positioning, rotation and scaling, but also quantification, text input and copy/pasting.

Each interaction technique can be implemented in a wide variety of ways, but they fundamentally rely on a natural and intuitive mapping between an action of the user and the resulting action in the virtual world and on an object. WIMP (windows, icons, mouse and pointer) is a successful 2D metaphor, but unfortunately a direct alternative is not available for VEs. An immersive environment requires several interaction techniques, because there is not one optimal interaction technique for all tasks [29].

2.3.1 Identification using an input device

Interaction with virtual objects implies that a relation needs to be determined with an input device and all virtual objects in the environment. When interaction is to be performed on objects, the objects first have to be identified before manipulation can take place. This is done using proximity detection algorithms that test for the proximity, collision or intersection of an input device with all virtual objects, achieving mouse and pointer functionality.

The time spent in testing intersections increases as the number of virtual objects increases. Typically, collision detection of an object is done hierarchically, by first intersecting a rough estimate of the object (usually a bounding volume) after which the intersected object is made more accurate. The intersection of an object with n faces typically has complexity $\mathcal{O}(n)$, but can be done in near $\mathcal{O}(1)$ when previous results are cached using the Voronoi Clip algorithm [17].

2.3.2 Object selection

Selection of an object is the first step in the complete interaction task. One first needs to identify an object before one can do any manipulation. All items in the virtual world must therefore be categorized as individual objects. The most natural form of object selection techniques come directly from the real world.

Direct grabbing (i.e. grabbing an object with the hand) is the most intuitive selection technique, but suffers from several drawbacks [25]. First of all, the object to be selected will most likely be obscured by the physical hand or input device. Although this is also the case in the real world, grabbing in the virtual world is still an estimation. Often only a static representation (e.g. a pointing finger) of the real hand is available. It is therefore not possible to feel the object or to place it perfectly in a virtual hand. This problem makes positioning and manipulation difficult.

Remote selection should be used to to acquire objects that are not in the user's physical range. Besides positioning oneself for direct grabbing using navigation techniques (such as flying, and walking; see section 2.4), there are also several techniques that allow selection without moving, such as arm-extension and ray-casting.

The efficiency of the selection task depends on several parameters. The size of the object

and the distance to the object are important. Generally a large, close object is easier to select than a small object farther away. However, due to perspective effects in immersive VR, large objects far away could be more difficult to select than a small one closeby [29]. The orientation to the object and object occlusion also influence the selection task. For example, selecting a square from the side will be extremely difficult. Far objects are also difficult to select when the tracking device is not accurate enough.

Arm-extension

Arm-extension techniques allow the arm to "grow" to the desired length so that remote objects can be grabbed. The manipulation step will become very intuitive because the user virtually has the object in the hand. Several variations of the arm-extension technique are common, which differ in the way that the user specifies the virtual arm length.

Bowman et al. describe the Go-Go arm-extension technique [4, 29] (named after the cartoon character Inspector Gadget). The mapping of the physical arm is split in two parts. The first part of the arm extension is linear to provide precise placement. The second part of the extension is non-linearly increasing so that objects too far away for the physical arm can also be reached. An advantage of this method is, that the user does not have to think about how to manipulate or grab an object. Selection of objects even farther away can be selected with some slight modification. The "fast Go-Go" technique has no local area and a more quickly growing extension function, and "stretch Go-Go" allows infinite arm extension.

A disadvantage of these arm extension techniques is that they can cause severe arm fatigue due to the continuous upholding of the arm. A variation that solves this problem is indirect arm-extension, where the use of two buttons enables extending or retracting the virtual arm. The natural mapping, however, is lost. Furthermore, placement of the hand within a remote object can become very difficult when the object is far away or when a small motion of the hand causes a large translation in the virtual hand.

Ray-casting

With ray-casting, the user's hand specifies the direction of a virtual ray of light used for selection [4, 5, 20, 22, 25]. Typically the ray is activated by pushing a button and an object is selected if the button is released while intersecting the object. Ray-casting is a direct analogy of the desktop 'point-and-click' method, which is so common (see figure 2.3).

Normally object selection is easy, even for objects that are farther away, because intersection is done on a direct line. This however means that a direct line of sight must exist between the user and the object. When tracker noise is high or when distant objects are too small to point at, problems can arise. Especially on distant objects these problems are amplified, because a small movement in hand orientation means a large movement in object position. In these cases, the cone-casting or spotlight variant, where not a ray but a cone emanates from the user's hand, solves some selection problems. In this case the diameter of the cone increases as the cone is farther from the hand.

Ray-casting is preferred for object selection [4], but is not very practical for manipulation because control of the object is not hand-centered. Objects are attached to the end of the

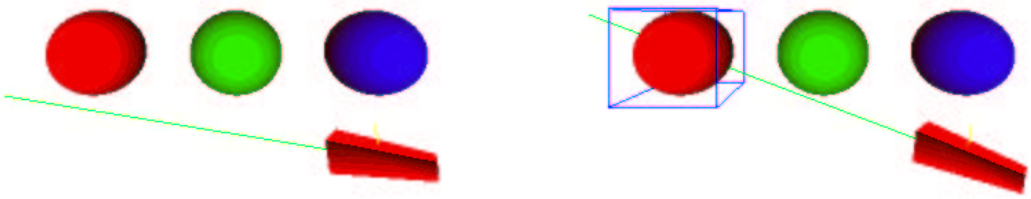


Figure 2.3: Selection of a virtual object with ray-casting, where the ray emanating from a manipulator like the wand does not intersect (left picture) and intersects (right picture) with a virtual object.

beam which makes rotation of the object difficult, except for around the axis of the ray itself. The only really usable interaction method is positioning the object by rotating or moving the hand. In this "stick" method, the object is fixed to wherever the ray of light intersected the object. Orientation and positioning of the wand allows one to orient the object in the sphere around the hand, using a one-to-one mapping of the hand position. Disadvantage is the limitation of movement without a series of grab-translate-release actions.

It is clear that ray-casting does not provide all six DOF. To increase the amount of freedom, Bowman et al. describe ray-casting with a fishing reel [4]. After selection, users can reel objects closer or farther away using two buttons, just like the indirect arm-stretching technique described above.

2.3.3 Object manipulation

Once an object is selected, actual "real work" or manipulation can be performed on that object. The term manipulation is actually a collection of many actions that change an object's property, including position, rotation, scale, color, etc.

Moving or positioning an object can be divided into three parts: the initial selection, large-scale movement and finally precise movement. Occlusion, visibility, initial distance to the object and direction to the object are important factors influencing the manipulation task [29].

Manipulation with the hand is most closely related to the real world, but this hand-centered manipulation can obscure the object to be manipulated. This problem can be solved by moving the center of rotation that is used in mapping rotations of the hand onto rotations of the object, to the center of the object resulting in object-centered manipulation.

As was the case with object selection, using a direct manipulation technique is often undesirable. Large scale objects are difficult the manipulate when held close to the user's head. Furthermore, a direct mapping also restricts the total amount of movement that is possible, making a series of grab-translate-release actions inevitable.

Constrained object manipulation

Manipulating an object in virtual space is often a direct one-to-one mapping of the movement of the hand. Like all the other tasks done with the hand, this unconstrained object manipulation means that doing precise work is difficult because the motor system of humans is often not precise enough for exact manipulation [20, 24]. Also, lag from software overhead, input devices and output devices can become a bottleneck [19]. To allow an interactive VE, a trade-off is often necessary between the accuracy of the system and speed [5].

Constraints limit the user's expressiveness, but as a consequence makes manipulation more precise. Lack of constraints when interacting with virtual objects is a major limitation in current virtual environments. Constraints are constantly used in the real-world and are useful in a virtual world. Allowing the user to choose between various interaction methods is favorable.

Constraints can be added by only allowing movement on a particular axis (or plane). The user will be able to move the hand in every direction, but every movement outside the plane will be restricted. These virtual constraints ignore extra degrees of freedom in the user's input [5, 20, 22]. Rotation can be virtually constrained by attaching control handles to the object on each principal axis. To rotate an object about one axis only, the user simply selects the appropriate handle. A slider widget can also be used to select direction of motion or rotation. However, these methods make the system more difficult to use and understand.

Object alignment can also constrain object manipulation. Snap-to-grid positioning or snap-to-orientation may provide enough constraints, but movement will become jerky and placing the object back at it's original position is difficult [22, 42]. Physical constraints are actual physical objects that are used to restrict the user's motion. An example of this is a desktop or hand-held paddle [44].

2.3.4 Multi-modal interaction

Different interaction techniques can be better suited, and often optimized, for specific interaction tasks. For example, ray-casting is easy for selection, but not for manipulation. Therefore, it is generally a good idea to make multiple interaction techniques available to the user.

Speech recognition technology is essentially a global input device, but can be made directional by combining it with another input device. These two modes together can function better than one alone [27]. Depending on the context of the system, identical voice commands can have different functionality. For example, a virtual menu only responding to voice selection commands when the user is looking at the menu.

Virtual menu

The low number of buttons on a controller frequently does not provide enough expressive power in complex virtual environments. To allow a more diverse command set, it can be useful to have a direct analogy of the two dimensional pull-down menu from common workstations available in VEs [5, 18, 20, 24]. Pull-down menus are widely used, mainly because there is a direct mapping between the user's movements of the mouse cursor and the menu, but also because selection of menu items is aided by the physical constraint of the table.

Due to the 2D nature of the menu, it can become difficult to locate or select a menu item from a menu, due to its position, scale and orientation with respect to the user. A frequently used method to avoid this problem is to lock virtual menus to a special point in the virtual space (world-fixed windows), or to have them float around in front of the user's view (view-fixed or display-fixed windows).

2.3.5 Quantitative input and output

An important virtual reality task is the input of a numerical value [20, 24, 29]. As stated above, manipulation done by moving the hand is not very accurate. Some manipulation techniques require the input of numbers, such as position offset or scale percentage. There are various ways these numbers can be entered. For example using virtual keypads or slider widgets on a virtual window [22]. Manipulation of widgets can use the same principles as described in section 2.3.2, but could also use specific physical devices, such as a "paddle" [18, 22].

The paddle is a physical tablet that the user holds in the non dominant hand, representing a virtual window in virtual space. Manipulations of widgets on the virtual window is done with the dominant hand. An advantage is that the user is being helped to position the hand because of passive haptic feedback. The use of proprioception makes it easy to use and allows for a large degree of precision. Downside is that it can lead to arm fatigue and it may be cumbersome to carry such a physical device.

In the real world, making notations of e.g. findings or reminders on a piece of paper is common. Only allowing input of a numerical value in the virtual world can therefore be restrictive. Annotation of an object in the virtual world with a string of characters can be performed with virtual keypads or physical devices that record handwriting. A more intuitive alternative is to use speech recognition technology (see chapter 3).

2.3.6 Measuring in VR

Once data has been visualized, quantitative information can be gathered from the presented data. This relatively new subject must allow a user to do measurements in VR. A geometric probing software architecture for interactive data exploration environments is GEOPROVE [1]. Geometric probes perform the measurements. By placing one or more markers, either the coordinates of a position in the environment or a value obtained from data at this position is obtained. A single marker is used to obtain position and value. Two markers are used to measure distance or derivative, etc.

2.4 Navigation

Full exploration of a large VE is only possible if the user is able to navigate through the environment. Often the VE is larger than the physical environment or beyond the tracking capabilities of the tracking system [21]. These large VEs require navigation techniques to allow exploration without any physical movement by the user. Mapping of physical motion

to virtual motion is one of the most intuitive means of movement through the virtual world [21]. This uses "proprioception": the ability to sense the location of body and limbs.

2.4.1 Overcoming the physical confinement of a VR system

Two of the most widely used navigation techniques are flying and walking. Besides controlling travelling speed one also needs to indicate the travelling direction by using a manipulator as a pointing indicator [20, 22, 25, 31]. The total freedom of movement provided by these 6-DOF pointing methods could lead to disorientation. Often pitch, roll and yaw are restricted resulting in 3-DOF flying. Walking constrains the user's motion further to 2-DOF, facilitating the user's ability to navigate the environment if moving in a 2D plane is adequate [21].

Flying and walking can be combined with a scaled world, where the world itself (or depending on the point of view: the user) is shrunk or enlarged. The user is then able to move to the desired location with only one step or make very delicate manipulations without making small movements, that will essentially eliminate tracker noise problems.

Mine et al. describe head-butt zoom, where a specific region of interest is framed with a rectangle [24]. When the user leans forward, and therefore positions the head inside the frame, the desired field-of-interest is zoomed in. Standing straight up again will return to the normal position. Advantage of this technique is that it uses proprioception and that rapid switching between different field-of-views is easy. Disadvantage can be placement of the frame and positioning oneself correctly in front of the frame.

There are also physical devices to control speed that map physical motion to the virtual world, such as a treadmill or ground-fixed bicycle. It is advisable to not only incorporate one technique, but to provide redundancy and allow the user to switch between various navigational methods. This however means additional complexity of the VE and requires some sort of switching method.

2.4.2 Wayfinding

As virtual worlds become very large, a user can quickly become disorientated while moving through the VE. Large VEs therefore offer "wayfinding" techniques which allows the user to determine his position. A distinction is made between three wayfinding tasks: naive search, primed search or exploration [10].

A naive (or exhaustive) search of the complete virtual world is commonly needed for first-time users who have no a priori knowledge of the location of a specific target. More advanced users, who do know certain targets, can do a primed search. Exploration is performed to gain insight in the VE without searching for a specific target. This "wandering" through the VE provides the user with important orientation independent survey information. Virtual maps, positional clues, such as 3D landmarks and paths between places of interest, increase the positional awareness [12].

2.4.3 Worlds-In-Miniature

A navigation technique that is also suitable for wayfinding is Worlds-In-Miniature (WIM). The WIM is a small representation of the total virtual environment, allowing global and local

context to be visible at the same time [20, 22, 36]. It is essentially an orientation dependent virtual map that can be used to instantly transport to another location. Besides these large scale movements, that facilitate easy navigation, also wayfinding tasks are easier. The user's own position and position of other objects are immediately clear.

Besides wayfinding and navigation, the WIM can also be used to select and manipulate objects through the iconic version if the VE. The user's range of interaction has effectively been extended by taking advantage of the ability to interact at multiple levels of scale simultaneously. Every object in the environment can be manipulated without having to move through the environment. Large scale movements are easy, but precise positioning and direct manipulation of objects can become difficult. Furthermore, having an extra miniature version of the virtual world can clutter the display.

2.5 Conclusion

In this chapter the hardware requirements of a Immersive Virtual Exploration Environments (IVEEs) and various interaction techniques (ITs), such as object selection, object manipulation and navigation, have been described in order to interact with objects in the environment. Conflicting as it may seem, these ITs often require constraints to make the interaction task easier.

Unfortunately, not every IT is best suited for every interaction task, which makes it necessary to have multiple ITs simultaneously available. However, more options make the system more difficult to understand. Multi-modal interaction will alleviate some difficulties by changing the behavior of techniques depending on the context of the system. Direct alternatives for the familiar WIMP (windows, icons, mouse, pointer) metaphor are the virtual menu and a 3D pointer, which makes intention of the user clear to the VR system by using an input device combined with tracking technology.

Sensory rendering provides important feedback from the VR system. The primary sense to be stimulated is vision, which must be rendered with an update rate of at least 10 Hz to avoid eye fatigue, impatience and manipulation problems of the user. Auditory rendering can be very effective to accentuate the interaction experience. Audio input through speech recognition is described in chapter 3 as an alternative and intuitive IT. Only recently, such speech recognition technology has matured to a point where reliable recognition can take place.

Chapter 4 describes a multi-modal interaction toolkit that incorporates the context sensitive speech recognition system from chapter 3. The primary selection and manipulation technique is performed with ray-casting. Rotation of virtual objects is accomplished using handles that constrain the rotation in only one principal axis.

Chapter 3

Context Sensitive Speech Recognition

Virtual scenes can become very large, which often result in low visual rendering update-rates. This will affect the user's ability to interact with rendered objects such as a virtual menu. A speech recognition system can operate completely separate of the rendering process. This makes it possible to give commands to environment efficiently even if visual rendering updates are low.

This chapter starts with an introduction to speech recognition technology and the various recognition errors that can be made by a recognition system. A context sensitive speech recognition library (i.e. CAVETalk) that has been created is described. By letting the system determine the context of the application one can increase the accuracy of the recognition system by minimalizing the vocabulary. This hypothesis will be scrutinized by experimentation.

3.1 Automatic Speech Recognition

Automatic Speech Recognition (ASR) involves the transformation of an acoustic speech signal into written text. Development of ASR has been ongoing for over 25 years and only recently there is coming widespread use for speech applications. Acceptance of speech technology is growing although there are still some key issues (i.e. accuracy and error management) holding it back. Speech recognition can be very useful in applications where hands and eyes are constantly busy, which is often the case in immersive environments, or for people with disabilities that inhibit their motor skills.

As speech recognition systems operate on a different input modality (namely sound), they are not hampered by limitations of graphical hard and software, thereby making it a suitable alternative interaction method in situations where visual complexity is always high.

3.1.1 Introduction

There are three categories of ASR systems: speaker independent, speaker dependent and adaptive. Speaker dependent speech recognition involves training for a specific user of the

recognition system. Each individual speaks differently, which can be contributed to a wide variety of reasons, such as accent or pitch.

Training the recognition system allows fine tuning to one particular voice. This is the main advantage and at the same time the main disadvantage. The advantage of training is that a very high word recognition accuracy can be obtained. Even with very large vocabularies (i.e. with tens of thousands of words), word error rates remain low. The disadvantage is, that every user must go through a time consuming training session. Another obvious disadvantage is the inability for others to use the same system without training.

For a lot of recognition systems it is often desirable that they can handle a wide variety of different, first-time or occasional, users (e.g. in automated telephone applications). ASR systems that have this extra degree of flexibility are called speaker independent. Generally, these systems are more difficult and expensive to develop and have lower accuracy then speaker dependent systems.

Adaptive systems lie somewhere between speaker independent and speaker dependent systems. An adaptive ASR system does not need explicit training. Instead, the system tries to learn the characteristics of the speaker as it is being used. This often requires that the user corrects the ASR systems in case words are misinterpreted.

Speech recognition systems that support dictation allow the speaker complete freedom in their choice of words. When only certain responses are allowed, the unconstrained choice of words in dictation can be limited to a set of predefined words from a vocabulary. As a tradeoff for freedom of speech, vocabulary based recognition will generally be more accurate.

Each of the three speech recognition systems categories can perform word recognition in four different strategies. Although certain strategies are common for either dictation or vocabulary based recognition, all strategies can be used for all different types.

discrete/isolated One word is recognized at a time. Separate words can only be detected if there is a small silence (e.g. 250 ms) between consecutive words. Speaking with pauses between words is very unnatural and often leads to overarticulation. Isolated word recognition is still widely used in many applications including dictation and telephone based conversational systems.

continuous/connected A series of words in a row, if separated by a silence of 50 ms or less, are recognized when they are in a predefined vocabulary. This type of recognition is very computationally expensive and requires some sort of artificial intelligence (AI) or validation technique for high accuracy.

phonetic Word recognition is based on the smallest distinguishable unit in a language, the phoneme. Words are defined as a string of phonemes, which can be represented as symbols in capital letters (e.g. aisles = AY L Z). This has the advantage that insertion of new words is easy, because only the phonetic representation of a word has to be added. The recognition task now consists of extracting the phonetic information from the spoken words and then use a best matching algorithm on the vocabulary. More accurate phonetic ASR systems use diphones or even triphones. A diphone is a pair of phonemes, that represent the transition between two phonemes. This automatically means recognition is more computationally expensive because more variations are possible. Triphones

bundle three phonemes together, but are rarely used.

spontaneous/conversational systems take all of the above to the extreme and allow the speaker to say anything. This would mean that the user can have a regular conversation with the computer. Because of the absolute freedom of speech, creating such a recognition system is extremely difficult. These systems try to figure out the context of the words in the sentence, just as humans do. Total freedom also implies that users are allowed to hesitate between words. The recognition task now consists of detecting key words in the sentence, which is not an easy task and very computationally expensive.

3.1.2 Performance metrics for ASR systems

There are several problems for ASR systems that make recognition difficult. At the basis are a wide variety of different languages and dialects. Words are almost always pronounced differently, contain hesitations and false starts [37, 47]. Lack of constraints leave too many possible sound variations for the recognition system to handle, which lead to errors [5]. The errors can be divided in the following categories [8, 46]:

deletion/rejection The recognition system fails to recognize an utterance (any stream of speech that represents a complete command) as a word from the vocabulary.

substitution The recognition system has recognized a particular word with great confidence, while the actual spoken word is different. This is a critical error, since there is no way to detect a substitution without verification.

insertion (or spurious recognition). A difficult error to detect and can be contributed to a number of causes. Often bad equipment or background noise can cause an insertion, but it can also be caused by the user thinking aloud or having a conversation with a colleague. Insertions can usually be kept to a minimum when it is possible to switch the ASR system off temporarily (e.g. with a mute button).

Every utterance processed by the ASR, can be classified in one of four categories. Either the recognized utterance is correct or it falls in one of the three error categories described above. Using these four categories, it is possible to give an indication of the accuracy of the speech recognition system. This accuracy is commonly referred to as the word error rate (WER) and can be defined as [40, 47]:

$$WER = \frac{substitution + deletion + insertion}{correct + substitution + deletion} \cdot 100\%.$$

3.2 Improving ASR performance by using application context

As stated above, the performance of speech recognition systems is still not optimal. This can be mainly contributed to the wide variety of possible responses from the user and the necessity of the recognition system to handle all of these responses. Allowing context sensitivity as a constraint may aid recognition accuracy.

3.2.1 Application context

Context sensitive speech recognition (CSSR) systems can have large vocabularies (i.e. thousands of words), but only a subset of that vocabulary will be activated at a particular time. For example, consider an application that supports a 'volume' command. In one particular context this command could refer to the output gain of an audio device while in another it could refer to a quantitative property of a three-dimensional object. In the context-insensitive case the user would have to add extra information (e.g. 'audio volume' or 'object volume') in order to unambiguously identify the target for this command. Although the recognition will perform better with this extra information added, the user is expected to have a thorough understanding of the ASR system's grammar, which will not make the system more user friendly.

By letting the system determine the context of the application one can increase the accuracy of the recognition system by minimalizing the vocabulary. Actual determination of the context is not a trivial task. Besides possible solutions such as pointing and gazing, one could also make use of Intelligent Agents (IAs) that execute autonomously, interfering minimally with the rest of the environment, apart from communication with other agents or the user [1]. Still subject to ongoing research at the University of Amsterdam, these IAs have the ability to perceive state changes through monitors, take actions that affect conditions in the environment and perform reasoning to interpret perceptions, solve problems and determine actions.

3.2.2 CAVETalk: a context switching speech recognition system

A speech recognition library capable of context sensitivity (CAVETalk) has been developed. The core of CAVETalk uses IBM's ViaVoice [15] as third-party recognition software. The engine of ViaVoice uses the Speech Manager API (SMAPI) native interface which runs under Linux. Currently, eight different languages (English, French, German, Italian, Spanish, Arabic, Japanese and Chinese) and several dialects of those languages are supported. Each dialect can be selected at run time.

ViaVoice is a speaker independent ASR system that can handle isolated or continuous recognition. User specific training is not necessary, but can be done to improve recognition accuracy. Although the ASR system supports multiple speech applications through a single engine, CAVETalk assumes that only one speech application is active at a time.

Vocabularies can be dynamic command vocabularies or structured grammars. Dynamic vocabularies are somewhat limited in functionality and are initialized on the fly. Structured grammars are an extension to dynamic vocabularies, which allow substitutions and repetition in grammar rules. A grammar is a structured collection of words and phrases bound together by rules, that define the set of all utterances that can be recognized by the engine at a given point in time. Each utterance is either accepted or rejected by the ASR system. The grammars are defined in a speech recognition control language (SRCL), which is a type of BNF (Backus-Naur Form) grammar especially designed for speech applications (see appendix A). The grammar compiler converts the BNF grammar file to a finite state grammar file (FSG). SRCL allows annotations to reduce the complexity of parsing command grammar sentences.

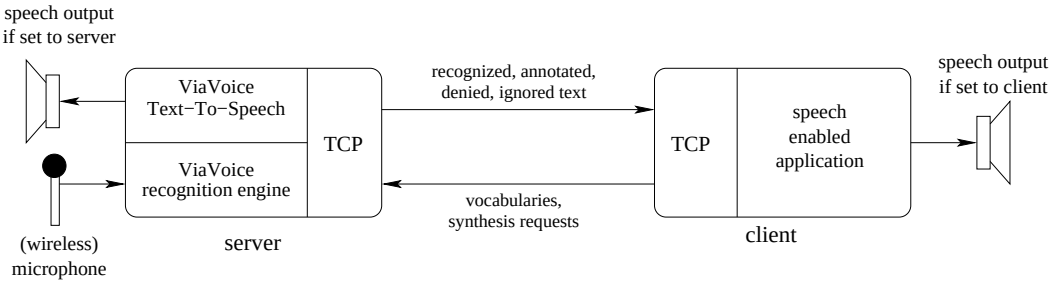


Figure 3.1: The architecture of CAVETalk and the communication between client and server.

3.2.3 Implementation

Communication between the speech recognition system (i.e. server) and the client (e.g. a CAVE application) is done via sockets communicating by TCP on a user defined port (7878 by default). It uses an easy to use C++ language socket library to handle most of the specific details concerning network communication [38]. Communication is always network byte-order (big-endian) and the protocol is based on sending messages between the client and the server (see figure 3.1). The server is a single-client server, which means that, every time a new client connects, the previous connection is terminated.

The client is able to send grammar files in SRCL BNF to the server, which compiles and prepares them. The client can then activate or deactivate one or more grammars at any time. The server monitors the input (from e.g. a microphone) and performs recognition based on the currently activated grammars. The client receives "annotated", "recognized", "denied" and "ignored" text through a callback from the server. The grammar that triggered the event is also included in the message. The microphone can be dynamically turned on and off by the client. This will further help a developer control when speech is enabled, without deactivating all the grammars.

Speech synthesis

ViaVoice TTS is used to add speech synthesis functionality to CAVETalk [15]. This library uses a platform-independent API, called the Eloquence Command Interface (ECI), to allow direct access to all of its functionality. ViaVoice TTS has the ability to synthesize to seven different languages (English, German, French, Spanish, Italian, Chinese, Portuguese) and several language dialects. Furthermore, numerous voice parameters can be changed, like pitch, speed, volume and speaking style. Output can be sent to three mutually exclusive destinations: a callback function, a file, or an audio device. The dictionaries used for synthesis can be customized to include unknown words, abbreviations, acronyms and other sequences. This customization is done with Symbolic Phonetic Representation (SPR).

The CAVETalk client is able to request synthesis of a specific sentence by sending a message. The server then synthesizes the sentence (in the digital audio format of either 8 kHz, 11 kHz, or 22 kHz) and either relays the result to the speaker of the server or to the client, which then handles the audio playback through a callback function. The choice between

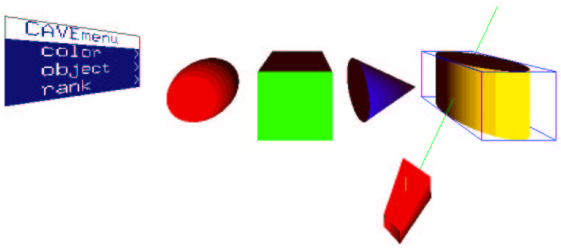


Figure 3.2: The four objects and menu in the CSSR experiment.

playback by the client or the server can be changed dynamically by the client. This way, a request by the client to play an arbitrary (non-speech) audio file is also possible by sending the audio file to the server if necessary.

3.3 Performance of CAVETalk

This user study evaluates if context sensitive speech recognition improves recognition accuracy (i.e. decreases WER) as opposed to conventional context insensitive speech recognition. Moreover, both context insensitive and context sensitive speech recognition methods will be compared to the more conventional (speechless) virtual menu.

The tasks performed in this test are not biased towards tasks which are a priori better suited for context sensitive speech recognition. In addition to accuracy, the completion time of the task is measured, giving an indication of the performance of each interaction method.

3.3.1 The experiment

Subjects

Several unpaid subjects were recruited from staff and students at the University of Amsterdam. The subjects had varying experience with the CAVE, from non-existent to near expert. All subjects were of Dutch nationality and the test was conducted using the American English version of ViaVoice. There were 8 male and 2 female subjects with varying pronunciation skills, from fluent to moderate (i.e. typical Dutch English). None of the subjects had any contact with one of the other subjects. Before performing the experiment, each subject was briefed shortly on the task to be done.

The task

The VR system used in the experiment is the CAVE located at SARA Amsterdam (see section 2.2.3). The virtual environment presented to each subject consists of four different objects: a sphere, a cube, a cone and a cylinder (see figure 3.2). Each object has a different color (resp. red, green, blue and yellow) and rank (resp. first to fourth). Therefore, each object has three distinct identifying properties: shape, color and rank. A virtual menu is

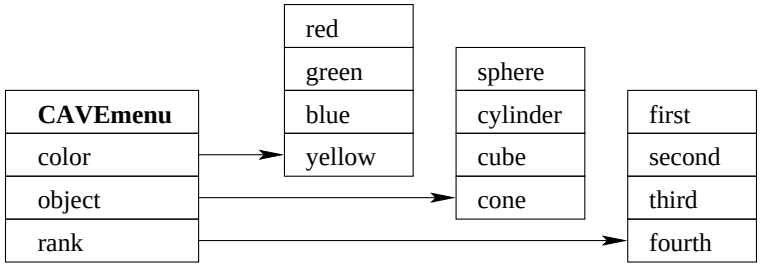


Figure 3.3: Structure of the SCAVI virtual menu used in the CSSR experiment.

placed on the left wall of the environment, containing three sub-menus to select between the three different properties (see figure 3.3). For each subject and throughout the complete experiment, the objects and the menu remain in the same position and retain their color.

The experiment is divided into three parts: (1) visual interaction with a hand tracked point-and-click device, (2) context insensitive speech interaction and (3) context sensitive speech interaction. Each part consists of ten property selection tasks (a total of 30 questions), based on a random computer generated question, that is given to the subject using speech synthesis. With three object properties there are six ($n(n-1)$) different types of questions that can be generated for each object (e.g. 'what color is the cone?' or 'what shape is the red object?'). Prior to the actual experiment the subjects are allowed to get accustomed to the tasks, by giving a small warm-up phase of two questions per part. Any last-minute problems, such as the required speaking volume, could then be addressed.

Each property selection task is alternated with an object selection task. This is done to return the subject's focus to the test objects. The reason for this is, that in some cases, the subject might be inclined to memorize all the objects and give a false sense of speed by only looking at the menu. Selection in the speechless part and the object selection task are performed with point-and-click interaction, which is frequently the most preferred IT [4]. The object selection task is done by selecting one of the objects in front of the subject, while the property selection in the speechless part is done through the virtual menu to the left of the objects.

The context insensitive speech part uses four different grammars: an object grammar (containing the words “sphere”, “cylinder”, “cube” and “cone”), a color grammar (containing the words “red”, “green”, “blue” and “yellow”), a rank grammar (with words “first”, “second”, “third” and “fourth”) and a global grammar with several object manipulation commands (such as “rotate 48 degrees” and “select sphere”). The first three grammars allow the subject to do property selection. The last grammar is added to mimic a true context insensitive speech enabled VE, which will almost always have certain commands available at all times (e.g. “quit”). A complete description of each grammar, used in the experiment, is presented in appendix A.

The context sensitive speech part only activates one of the first three grammars that are needed to complete the property selection task. The fourth grammar is always activated. The virtual menu is hidden in both speech parts, to give a visual clue that commands must be issued using speech.

user says	*recognition system*	*recognition error*
command x	accepts command x	correct
command x	accepts command y	substitution of command y
command x	does not accept	deletion
no command	accepts command x	insertion of command x
no command	does not accept	deletion

Table 3.1: Overview of all possible recognition errors made by the recognition system.

Verification

All steps performed by the subject are logged including the precise moment in time the action occurred. The first part of the experiment, i.e. menu selection with a point-and-click device, and the object selection task are validated automatically. In these cases, the system will wait when an incorrect answer is given, and continue with the next task when a correct answer is given.

The ViaVoice recognition system classifies each utterance in two categories: accepted and unaccepted. Table 3.1 gives an overview of all possible recognition errors that the recognition system can make. All unaccepted commands are immediately correctly classified as a deletion, but all accepted commands are not by definition correctly classified. Therefore, automatic validation is impossible as the table 3.2 illustrates.

An automatic validation system suffers from two unrecoverable classifications as falsely correct. First, when the user speaks the wrong command, but a substitution of the expected command occurred. Second, when the user does not say a command (e.g. he talks to a colleague), but an insertion of the expected command occurred. This means that every command the recognition system classifies as correct, including user commands that are not expected, must be manually verified. The object selection task incidentally gives the verifier enough time to immediately validate each command and note down what the user has been saying. Besides speaking commands, a user may also be inclined to laugh, speak to the operator, hesitate or give a wrong command. In both speech parts, the system will only continue with

test expects	*user says*	*recognition error*	*classification*
command x	command x	correct	correct
command x	command x	substitution, deletion	incorrect
command x	command y	substitution of command x	**falsely correct**
command x	command y	substitution of command z (z≠x), deletion, correct	incorrect
command x	no command	insertion of command x	**falsely correct**
command x	no command	insertion of command y, deletion	incorrect

Table 3.2: Overview of all possible classifications in an automatic utterance validation system (unrecoverable problems are in boldface).

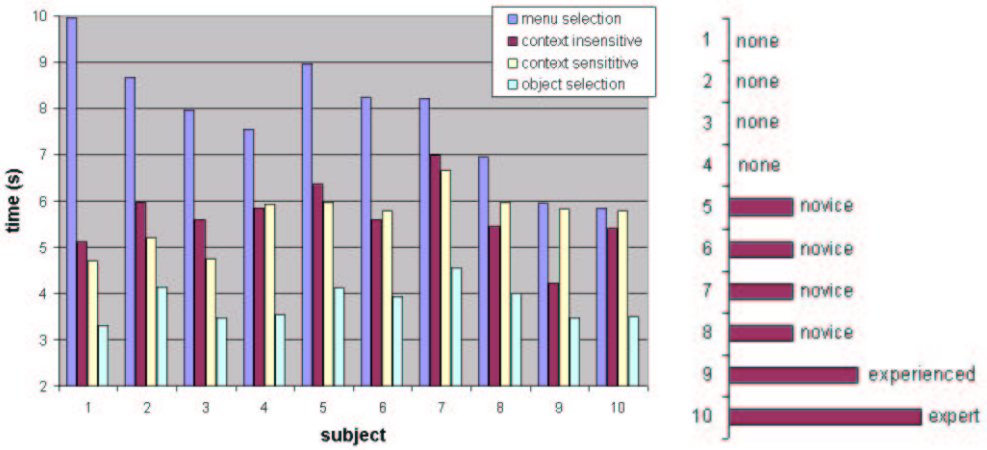


Figure 3.4: Average task completion time (including the time needed to ask the question) and CAVE experience of each subject.

the next task when the verifier determined if a valid command was given (i.e. by pressing a button).

3.3.2 Results

One of the first results that can be obtained from the experiment, is the average time with which subjects are able to complete a specific task. Figure 3.4 shows the average time taken per task and the amount of experience each subject had with the CAVE. A first-time user has never been in the CAVE. A novice user has been in the CAVE, but never used a manipulator such as the wand. Experienced and expert users have had quite a lot more experience with pointing and manipulation in the CAVE.

Figure 3.4 shows that object selection is performed very fast and, although there are some fluctuations, there is not a real distinction between experienced and first-time users. Menu selection is clearly the slowest method to select an object especially for novice and first-time users. Selection of objects using speech (context sensitive and context insensitive) is faster then menu selection. There is no real difference in completion time between the context sensitive and context insensitive speech tasks.

The Word Error Rates (WERs) presented in figure 3.5 indicate that context sensitive speech recognition is not as accurate as context insensitive speech recognition. The only exceptions are subjects 8 and 10, who had a perfect speech session without any errors, and subject 6 who performed better in the context sensitive part although the results for subjects 3 and 6 cancel each other out.

3.3.3 Discussion

Considering the small size of the experiment and the number of subjects, the results only give a modest impression of the facts. Menu selection is typically slower for inexperienced

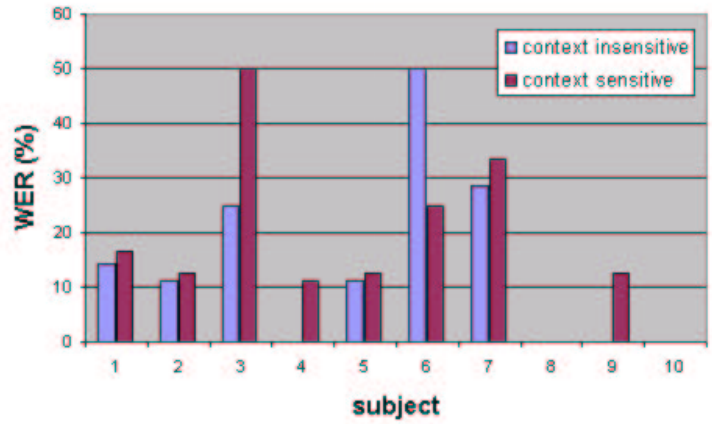


Figure 3.5: The Word Error Rate (WER) of each subject for the context sensitive and context insensitive tasks.

subjects, which can be contributed to lack of experience with using the wand. First-time users do not have the same degree of positioning control of the ray emanating from the wand. The time taken by context sensitive and context insensitive speech recognition does not differ that much and thus no real performance gain time-wise can be established.

It is quite striking to see that the totally inexperienced subjects have relatively faster (i.e. about 1 second) speech completion time with respect to the more experienced subjects. This could be caused by the fact that more experienced subjects have used speech recognition software before and therefore know that the recognition is more accurate when it is quiet. These subjects will then wait until the entire question has been stated and could unintentionally give the recognition system some time to get ready.

Because not all of the subjects were fluent English speakers, there were quite a lot of word pronunciation problems (see figure 3.6). The American English recognition system was therefore unable to correctly recognize those poorly pronounced words, such as “cone”, “first”, “red” and “third”. Proportionally, subject 6 was required to answer much more problematic words in the context insensitive part. When figures 3.5 and 3.6 are compared, one can see a direct relation between the WER and the number of pronunciation problems, making the comparison between context sensitive and context insensitive speech recognition using WER unfair. This problem could have been diminished by not randomizing the questions for each subject, but to keep the questions fixed. Especially for such a small number of experiments per part, it might have been better to manually determine an optimal question-set to remove question repetition. Increasing the number of questions or repeating sessions will also makes it possible to determine standard deviations. However, only one session was performed due to the large amount of time needed for each experiment.

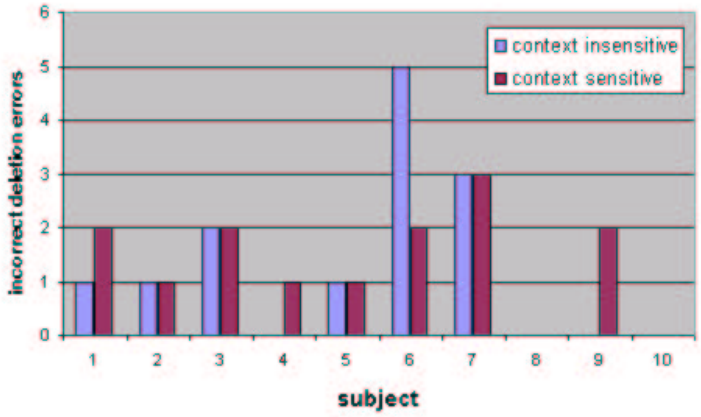


Figure 3.6: The number of incorrect deletion errors for each subject due to typical Dutch English.

3.4 Conclusion

The context sensitive speech recognition system (i.e. CAVETalk) presented in this chapter provides a developer with a toolkit that makes it possible to dynamically activate different vocabularies for a specific application context. This has been demonstrated in an experiment that validates the performance of CAVETalk. In situations where visual complexity is always high and interaction with the virtual environment is difficult due to a low update rate, speech is very helpful because it operates on a different modality. The ViaVoice recognition system is accurate when the speaker is a fluent in American English, making it very realistic to incorporate ViaVoice in several serious applications. CAVETalk is an extra layer on top of the ViaVoice recognition system making it as accurate as ViaVoice, but without the complex interface of ViaVoice. A developer can therefore quickly create a speech enabled application.

Results from experimentation with CAVETalk have shown that object selection using speech input is very fast and provides a natural interface, especially compared to the more traditional virtual menu. Enabling only a subset of words for a particular context did not show any recognition improvements over enabling all words. One of the major limiting factors for a definitive conclusion are the small number of experiments. Therefore, context sensitive speech recognition may still proof beneficial in future applications. This is an area of future research. The next chapter includes the presented CAVETalk context switching speech recognition system in a multi-modal interaction toolkit.

Chapter 4

SCAVI: a multi-modal interaction toolkit

4.1 Introduction

Many virtual reality applications are restricted to walkthroughs. However, walking is not sufficient to do actual work in a VE. To really interact with a VE one must be able to perform direct object manipulation which is not an easy task (see section 2.3). Interaction methods that facilitate such manipulation in VR are relatively underdeveloped. This chapter presents a user interface library for immersive environments, the SCAVI toolkit.

4.1.1 Previous work: the Virtual Radiology Explorer

The Virtual Radiology Explorer (VRE) is a static exploration environment capable of visualizing medial CT and/or MRI data [2]. The VRE allows medical data from hospitals to be processed on remote high performance computing (HPC) systems for 3D visualization. VRE uses several libraries such as CAVELib and the Visualization Toolkit (Vtk) for visualization in the CAVE. VRE does not include direct object interaction techniques. The main impulse for the SCAVI toolkit is to provide the VRE with direct object manipulation techniques.

The CAVE library (CAVELib)

CAVELib is an Application Programmers Interface (API) designed for developing virtual reality applications for spatially immersive displays [43]. CAVELib has been in development since the debut of the CAVE in 1992, and is widely used in research and academic areas.

The current implementation supports up to 32 tracking sensors and several controllers consisting of buttons and/or valuators, such as a joystick. The controllers provide floating point values, usually normalized between -1 and 1. The standard controller used by CAVELib is the "wand", which is a 3D tracked mouse with several buttons and a joystick.

To be able to parallelize the VR system, CAVELib forks separate processes for each display (i.e. CAVE wall), providing each process with their own memory space. Communication between processes is done using shared memory, which can result in problems if no mutual

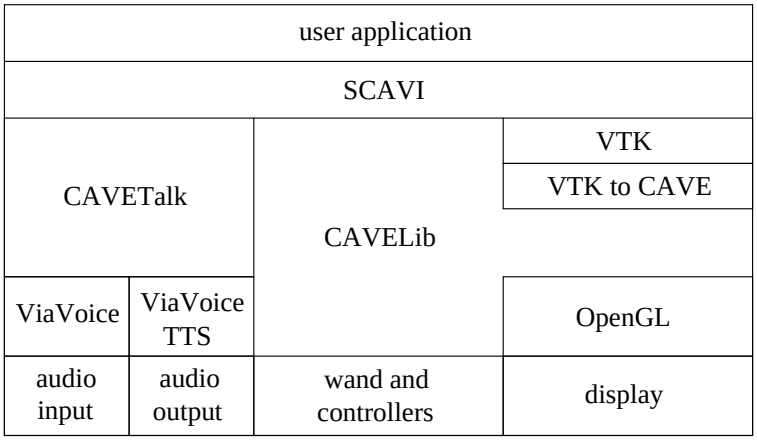


Figure 4.1: Schematic overview of the SCAVI architecture.

exclusion is used to synchronize processes. The states of the trackers and controllers are also stored in shared memory. Several library functions allow a developer to access this data.

Each display process is synchronized and automatically renders the correct perspective for a wall. The library also has functions for virtual environment sharing over a network. Flexible configuration files make programs written with CAVELib portable to several display and input devices without the need to recompile. CAVELib is currently available for IRIX and Linux.

The Visualization Toolkit (Vtk)

Vtk is an open source software system for 3D computer graphics, image processing and visualization [34]. A visualization pipeline transforms numerical data into geometric constructs that are rendered into visual presentations. Vtk includes a C++ class library containing over 500 visualization objects.

The pipeline starts with source objects, that can be dynamically created by Vtk functions or objects from external sources, such as simulations or CAD programs. A source object can go through several mappers that alter the geometry. An "actor" object is used to actually visualize a virtual object. Other examples of objects are light sources, cameras and mathematical functions. The benefit of Vtk is that it is flexible, extensible and runs on many computing platforms (nearly all Unix-based systems and PCs).

Previous work by Matthew Hall provides a way to incorporate Vtk objects in a CAVELib application with vtk2CAVE [13]. All Vtk actors are copied in shared memory to make them accessible to all display processes. Only Vtk polygonal data can be visualized by translation to OpenGL. Dynamically changing data is handled automatically. Not all of Vtk's functionality is supported, most notably cameras and lights are unsupported. Several extensions to the original version have been made that include actor opacity, removal, visibility and texturing.

4.1.2 The SCAVI interaction toolkit

The work presented here is directed towards extending the VRE with interaction methods. Therefore, it uses Vtk and CAVELib as an integral part (see figure 4.1).

The acronym SCAVI of the toolkit stands for "Speech, CAVE and Vtk Interaction". The current implementation has a strong tendency towards the CAVE (see section 2.2.3) because it was the only VR system available. Therefore, the assumption is made that the input device of the VR system is a "wand", a 3D mouse with three buttons and a joystick. The implementation of interaction techniques in this chapter reflects this.

Development of CAVE applications is done with VRCO's CAVELib [43], which makes use of SGI's OpenGL [45] or OpenGL Performer [30]. CAVELib has general support for every conceivable input device making it possible to adjust the current implementation of SCAVI for other input devices with similar functionality. Conversion to another VR system is far more difficult, because SCAVI relies on the ability to render Vtk objects in the environment using CAVELib. When another VR system has this capability, it is not very hard to adjust the SCAVI toolkit for that system.

4.1.3 Related work

Bowman et al. present the Conceptual Design Space (CDS) offering a complex system of tools for architectural design [5]. Special attention is given to constraints. A fixed set of primitives can be created and grouped together to form more complex structures.

The HOMER (Hand-centered Object Manipulation Extending Ray-casting) system also by Bowman et al. offers object selection via ray-casting [4]. After selection, manipulation is done by extending the hand to the center of the object, such that hand-centered manipulation becomes possible. This will make selection of remote objects easy and with minimal effort.

Mark Mine presents an Immersive Simulation Animation And Construction (ISAAC) for the construction of virtual worlds using direct and indirect manipulation techniques [20]. ISAAC is not a modelling program, it uses pre-generated 3D models from several sources, such as CAD programs.

Mark Mine has also presented the Chapel Hill Immersive Modelling Program modelling program (CHIMP) for the preliminary phases of architectural design [22, 23]. It uses both one handed and two handed interaction techniques for object-centered manipulation with cone-casting and incorporates a look-at menu.

Unfortunately, the libraries these studies produced are only applicable for specific tasks. A different approach has been proposed by Michael Rorke et al. [32]. They present an interaction abstraction framework, where an interaction component communicates directly with all the objects in the system. Dependencies of the objects are handled by the objects themselves. Therefore interaction requests are relayed to the object instead of handled by the interaction component. The implementation in the next section adopts this framework, with one difference. The object dependencies are not handled by the object, but by the interaction component.

4.2 Design of SCAVI

Because the design of SCAVI uses vtk2CAVE as one of the first layers, the structure of a developer's program is also closely related to the structure of a vtk2CAVE program. The user calls a specific SCAVI draw function (i.e. called for each display process) and a SCAVI loop function that handles the interaction, such as object placement and menu selection.

4.2.1 SCAVI objects

Interacting with objects can only be done if there are distinct objects that can be identified. Every object in SCAVI is a Vtk actor with several enhancements defined in the C++ class called vtkCAVEActor. All SCAVI objects are kept in a dynamically linked list (DLL), which is not kept in shared memory making it only available to the main CAVE process and none of the display processes. The main extra features of a SCAVI object over a Vtk actor is the ability to add several event handlers to an object and the ability of the interaction component to identify and perform interaction on an object. A SCAVI object can also be created as a regular Vtk actor, without it being handled by SCAVI.

Each SCAVI object can be given a name so that voice identification using speech technology or enumeration in a virtual menu becomes possible. Besides regular Vtk functionality, such as changing the color of the object or the visibility state, several functions have been rewritten to accommodate for the extended functionality, such as delete and copy. The rotating, scaling and translation functions are also rewritten, which makes it possible to transparently use either the transformation matrix at the end of the Vtk pipeline (i.e. from the actor) or after the source input. The benefit of using a transformation matrix directly after the source input is that use of subsequent Vtk filters will be aware of changes in the object's position, scale and orientation.

Events sent by SCAVI objects

The developer's program receives several events when certain actions have been performed on a SCAVI object. A "focus" event is sent by SCAVI when an object has attracted or lost focus. The "button" and "joystick" events are sent when the corresponding valuators on the wand have changed.

These three events are sufficient and necessary to form more complex tasks. For example, there is no explicit movement event, but can be created with the focus and button events. Movement of an object begins with a focus event followed by a button pressed event. The movement will end when the button is released, making it possible for a developer to determine the displacement of the object.

The fourth event currently implemented is a speech event, which is called when the user directs a voice command towards a SCAVI object. Other input devices can also be included to SCAVI by adding new event handlers (e.g. for a data glove).

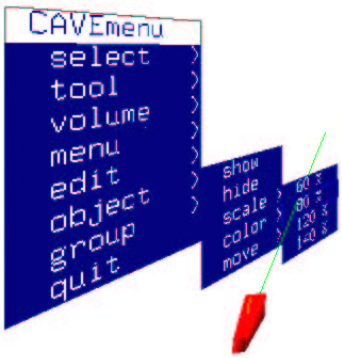


Figure 4.2: The two dimensional virtual menu in the SCAVI environment.

4.2.2 Two dimensional virtual menu

Virtual menus are required to be clearly visible, have simple interaction techniques, have an efficient menu structure and account for both novice and expert use [11]. Care must be taken not to obscure much of the user's view of the VE. The two dimensional menu items can become difficult to locate or select, due to its position, scale and orientation with respect to the user.

Implementation

The vtkCAVEMenu class allows several 2D virtual menus to be added to the environment. Each virtual menu is a subclass of vtkCAVEActor and each menu object is also a SCAVI object with the menu as their parent. The menu is a hierarchically structured list of menu objects consisting of on/off buttons, items or sub-menus. Each sub-menu resides, like the SCAVI objects list, in a DLL outside shared memory. The complete menu structure is constructed on the main menu DLL.

Two callback routines can be defined by the developer. The first is a callback on selection of a menu item, making it possible to conduct an appropriate response. The second is an expand callback, which is sent just prior to the display of a sub-menu. This makes it possible for the developer to dynamically change the content of a sub-menu (e.g. list all currently present SCAVI object names).

Although one of the requirements for an effective virtual menu is that characters are clearly visible, the current implementation only uses fixed size characters of 15 points (see figure 4.2). In certain cases this makes it difficult for the user to read the menu at large distances.

Currently, one menu can implicitly and automatically be made voice enabled, as will be described in section 4.2.3.

4.2.3 Audio I/O

SCAVI makes use of the separately developed CAVETalk context sensitive speech recognition library discussed in chapter 3 for audio input and output (I/O). Voice commands can

be directed towards specific SCAVI objects, that will then receive an event, or have a global context allowing a global callback routine to handle the speech event. Any number of different grammars can be dynamically attached to several SCAVI objects. All functionality is defined in vtkCAVESpeech, including a special function that makes it possible to repeat the last voice command.

Since complex scenes result in very poor framerates, making interaction with a menu difficult, it is a valuable asset to allow voice selection of menu items. The virtual menu of section 4.2.2 is implicitly speech enabled, but this can be enabled and disabled at run time. The contents of a sub-menu are exported to a BNF grammar file so that it is automatically recognized by the CAVETalk speech recognition library described in section 3.2.2. These automatically generated grammar files are constructed such that the CAVETalk server, besides the recognized text, returns an annotation of the unique SCAVI object (i.e. menu item) number that has been voice selected.

4.2.4 Editing in structured point datasets

Medical scanners and computer simulations generally produce very large structured point datasets. Because it is sometimes desirable to make large or small modifications to these datasets (e.g. to remove artifacts from a CT scan), support for such modifications has been included in SCAVI using the vtkCAVEVolume class. The third button on the wand can be used to switch between a editing tool and the default ray-casting tool. Activation of either tools can be changed dynamically by the developer's program.

Editing is the task of changing the value of data elements in a structured point dataset (or structured grid). First, the structured grid is read by a SCAVI volume actor, which, like the menu actor, is an extension to the regular SCAVI actor. The toolkit then visualizes the volume by rendering the contour as triangle strips. A volume can only be edited when it is first selected. Editing is done while the first wand button is pressed. Currently only a cubical brush is implemented, the size and color of which can be modified by a menu option or via a voice-command. A transparent brush acts like an eraser.

Large volumes may still be too computationally expensive to render efficiently as a contour. The complexity of the structured grid can, therefore, be decreased with sub-sampling. However, such sub-sampling can introduce serious topological changes to the dataset (e.g. see figure 4.5). Viewing the volume as a contour (i.e. triangle strips) has another disadvantage, because the editor cannot see the base-level grid points anymore. Displaying all grid points as voxels (volumetric pixels) can, however, result in a very low update frequency. A hybrid of voxel and contour viewing provides an intermediate solution to this problem. The second button on the wand allows the user to select a cubical region of the selected volume which has to be visualized as separate voxels. This region is commonly referred to as the volume of interest (VOI). In order to be able to distinguish individual voxels, their size occupies only about three quarters of the actual space. Furthermore, each voxel is partly transparent making it possible to see underlying voxels (see figure 4.3).

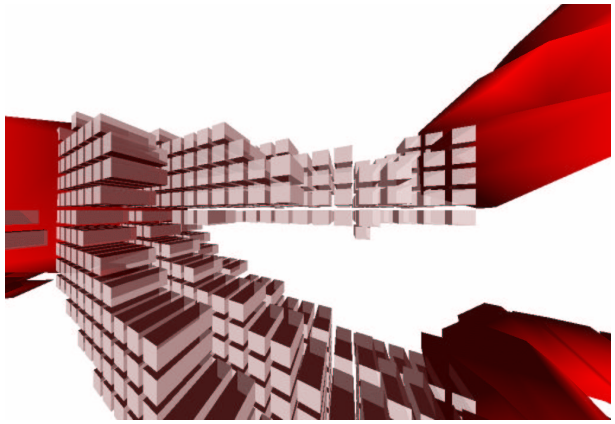


Figure 4.3: Representation of voxels as partly transparent cubes.

4.2.5 The central interaction component

The components of the previous four sections (i.e. objects, menu, audio and volumes) are tied together by the central interaction component defined in the vtkCAVEInteraction class. The structure of each class is modelled after the typical Vtk class structure, making it very easy for a developer to use SCAVI when they are familiar with Vtk.

The wand input device used during development contains three buttons and a joystick. There are several tools that can be cycled through using the third wand button. Currently supported tools are selection and manipulation through respectively ray-casting and editing.

A loop function, which must be called frequently by the developer, monitors the state of the valuators (i.e. buttons and joystick) on the wand. When one of the valuators has changed, appropriate events are sent to the SCAVI object in focus and used in the internal interaction routines.

The speech client and the menu can be attached when starting the SCAVI application, which will make it clear to the central interaction component that the specific feature must be made available. Speech support is enabled when a valid connection could be established to the CAVETalk server. The virtual menu will then automatically be speech enabled by sending a grammar file of the current (sub-)menu to the server.

The developer has the ability to cycle through all the SCAVI objects from the DLL, making it possible to do manipulations on objects through e.g. voice and menu commands instead of using direct manipulation using the wand. But also making it possible to enumerate the names of all SCAVI objects e.g. for display in the virtual menu or to synthesize audible messages (e.g. “there are 2 cubes and 1 sphere”).

Selection through ray-casting

The ray-casting tool is used to identify an object from a set of objects, because it is the most preferred and intuitive WIMP-like selection interaction technique. The actual intersection test of the ray with the object, performed on all objects in the SCAVI DLL, is done with two functions that are part of the Vtk library. The first is a line intersection test with a Vtk

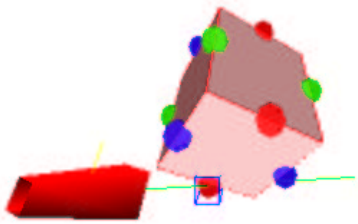


Figure 4.4: Rotating an object with SCAVI rotational handles.

object, and the second is function that calculates the distance between two points such that the closest object can be identified. Curiously enough, since the Vtk intersection test is done hierarchically, the intersection algorithm can be made faster by first checking if the line of the ray intersects the bounding-box of the object. Therefore, this faster method has been implemented in SCAVI.

Only SCAVI objects that are visible and pickable can become focussed (triggering a focus event) whenever the ray has a direct line-of-sight to that object. A focussed object can be grabbed by holding the first wand button pressed. Pressing the second wand button on a focussed object selects that object, such that more elaborate manipulation can be performed. Currently only one SCAVI object can be selected at a time. The developer can request a pointer to the selected object.

Manipulation of SCAVI objects

Once an object is selected, twelve rotational handles are attached to the edges of the object's bounding-box (see figure 4.4), adding constraints for accurate object rotation. These handles are similar to regular SCAVI objects, but they can only be moved with the selection method described above. Four handles are attached for each principal axis about which rotation can take place. The handles are colored red, green and blue for the X, Y and Z axes respectively.

A grabbed object can be moved by moving or rotating the wand. Scaling can be performed by moving the joystick up (enlarging) or down (shrinking) while the left button is pressed on the object. As with all joystick operations where the values may "drift", the joystick must be moved beyond a threshold level before any action takes place. Once a SCAVI object is selected, voice commands can be directed to it without the object being in focus.

Feedback

Besides the immediate feedback the user receives from the VE by changing the position and orientation of the head, the interaction component must also make the user aware what is happening. Movement of the wand with ray-casting will immediately show changes in the position and orientation of the ray. Further feedback by the system includes showing the bounding box of a focussed (painted blue) or selected (painted red) object. The bounding box

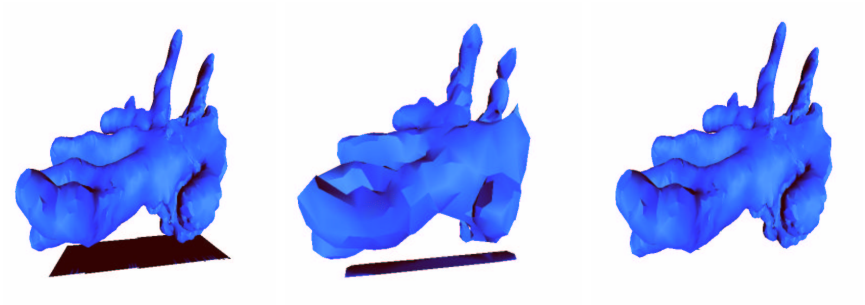


Figure 4.5: Contour representation of the 512x512x183 coral dataset with no sub-sampling consisting of 62646 triangles (left), factor 10 sub-sampling with 4520 triangles (middle). The right picture shows the coral without the table, which has been removed with the SCAVI editing tool.

information is put in shared memory by the main process, making it available to the display processes.

Editing is made clear to the user by a short ray with the bounding box of the currently selected pen attached. This bounding box is only visible when the end of the editing ray is within the boundaries of the volume.

Auditory feedback is not implicitly woven in the SCAVI toolkit. Audio synthesis and other audio output such as short sounds or loops are left to the application developer to define.

4.3 Test case: removing artifacts from a CT scan

A medical scanning technique such as a computed tomography (CT) will generally produce a three dimensional dataset that contains more information than was intended. Several researchers at the University of Amsterdam have special interest in corals scanned with CT, which are being produced with increasing accuracy and detail. Before further research such as simulations can be performed on these coral datasets, unwanted artifacts such as the table on which the coral is lying and rubber bands holding pieces together must be removed. The traditional process of manually reviewing and editing each 2D slice is very labour intensive. Furthermore, the relation between successive slices is very difficult to determine.

4.3.1 The experiment

An immersive environment should be ideal to visualize these fundamentally 3D coral datasets. This test case uses the SCAVI editing interaction techniques described in section 4.2.4 to make the removal of artifacts more intuitive. The main focus of this test case is to determine the usability and effectiveness of the interaction toolkit and whether the removal of the table from a 512x512x183 coral dataset (see left picture of figure 4.5) can be done efficiently. One of the main reasons to use a coral dataset is because of its high complexity. Effective editing can only be performed when the user is able to efficiently interact with the VE. Therefore, one first needs to determine whether the coral dataset can be visualized at sufficient speed.

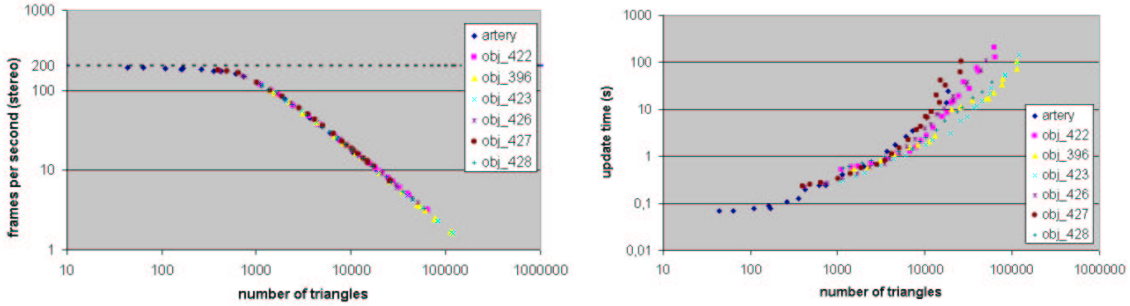


Figure 4.6: Number of rendered frames per second (left) and the time required for the recalculation of a contour (right) in relation to the number of triangles of several coral objects and one abdominal artery scan.

The VR system used is the Linux Immersive Environment (LIE) as described by Belleman et al. in [3], which is built around a commodity computer using a commodity graphics adapter and running the Linux operating system (i.e. dual Pentium III at 1 Ghz with 1 Gb RAM, GeForce2 Ultra DDR and RedHat Linux kernel 2.4.5). Belleman et al. have determined that graphics performance on such a system which uses commodity off-the-shelf computing and graphics hardware (i.e. LIE) is of the same order as high-end VR systems (e.g. CAVE). The speed results obtained using the LIE are therefore a good indication of the speed in a high-end VR system.

4.3.2 Results

The visualization speed (i.e. updates per second) of the VR system is directly dependent on the number of triangles that are rendered by the environment (see the left picture of figure 4.6). There are two factors that influence the amount of triangles in the environment.

First, increasing the contour sub-sampling factor decreases the amount of points in the dataset and therefore decreases the amount of triangles in the contour, but as a consequence will make the visualization increasingly less accurate (see the second picture of figure 4.5). Cropping the dataset will similarly decrease the amount of triangles, but a complete overview of the dataset is lost.

Second, the amount of visualized voxels influences the number of triangles, because each voxel is constructed out of between 0 and 12 triangles. It is therefore beneficial for the visualization speed to increase the voxel sub-sampling rate and/or to keep the number of voxels to a minimum by keeping the VOI as small as possible.

The manufacturer of the 3D graphics accelerator (i.e. GeForce2 Ultra DDR) claims that their hardware can render 10 million triangles per second. The left picture of figure 4.5 shows that this is not nearly the case, which can be contributed due to large overhead by CAVELib, vtk2CAVE and Vtk. The picture also shows that the number of updates per second do not constantly increase with fewer triangles in the scene. The update rate does not exceed 200 Hz, which is probably due to limited bandwidth on the motherboard.

The contour and the voxels are recalculated when a region in the dataset has been edited

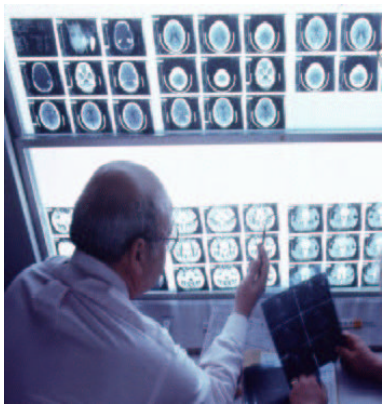


Figure 4.7: Diagnosis through two dimensional medical images on a light-box.

or when the VOI has been altered. While recalculating, interaction with the environment is compromised making it necessary to minimize the time spent in recalculation. The right picture of figure 4.6 shows the time needed for the recalculation of a contour compared to the size of the constructed contour. Decreasing the sub-sampling rate of a contour will increase the recalculation speed exponentially.

Section 2.2.1 suggested a minimal update frequency of 10 Hz, which can be achieved with the coral dataset when a sub-sampling factor of 2 is used. However, this means that the recalculation of the contour takes about 40 seconds, which is too high for practical interaction unless the user is informed on the status of the calculation through e.g. a progress indicator. A usable recalculation time is about 3 seconds that can be achieved when a sub-sampling factor of 6 is used. The resulting contour of about 10.000 triangles can then also be efficiently visualized at 32 Hz while retaining reasonable visual realism. The removal of the table from a coral dataset using a sub-sampling factor of 6 was performed in the LIE in a few minutes. The result is presented in the right picture of figure 4.5.

4.4 Test case: vascular reconstruction in a virtual operating theater

Scientific progress in medicine has been enormous over the last few decades. Today, medical scanning techniques, like CT (computed tomography) and MRI (magnetic resonance imaging), allow a surgeon to non-invasively look inside a patient's body to diagnose an illness. Unlike with X-ray, these scanners are able to create images of numerous cross-sections of the patient's body. To form a diagnosis, the surgeon or a researcher can then look at the slices on a light-box (figure 4.7). This is not always an easy task, since the surgeon must try to make a mental three dimensional reconstruction of all the two dimensional slices combined.

The combination task can be aided with the help of applications that allow the user (e.g. the radiologist or surgeon) to view and rotate all of the slices stacked onto each other by computer [14, 26, 34]. Additionally, such visualization techniques also allows one to do many

other complex tasks that are not possible through conventional methods. For example, one could let the application remove all of the skin from the visualization or to only show the bone structures, allowing the user to focus on a specific part without any obstruction.

This section describes a research effort currently conducted at the Section Computational Science of the University of Amsterdam in which a virtual vascular reconstruction operating theater is developed[1]. In this system, the SCAVI interaction techniques developed in chapter 4 are used. The theater is a VR visualization program, which gives a surgeon or researcher a set of manipulation tools that will make it possible to plan, rehearse or validate certain surgical procedures in an intuitive way.

4.4.1 Vascular disorders

Vascular disorders in general fall into two categories: *stenosis*, a constriction or narrowing of the artery by the buildup over time of fat, cholesterol and other substances in the vascular wall, and *aneurysms*, a ballooning-out of the wall of an artery, vein or the heart due to weakening of the wall. Aneurysms are often caused or aggravated by high blood pressure or wall shear stress. They are not always life-threatening, but serious consequences can result if one bursts.

A vascular disorder can be detected by several imaging techniques such as X-ray angiography, MRI (magnetic resonance imaging) or computed tomography (CT). Magnetic resonance angiography (MRA) has excited the interest of many physicians working in cardiovascular disease because of its ability to non-invasively visualize vascular disease. Its potential to replace conventional X-ray angiography that uses iodinated contrast has been recognized for many years, and this interest has been stimulated by the current emphasis on cost containment, outpatient evaluation, and minimally invasive diagnosis and therapy.

A surgeon may decide on different treatments in different circumstances and on different occasions but all these treatments aim to improve the blood flow of the affected area. Common options include thrombolysis, balloon angioplasty, stent placement or vascular surgery. A surgeon resorts to vascular surgery when less invasive treatments are unavailable. In the case of an *endarterectomy* the surgeon opens the artery to remove plaque buildup in the affected areas. In vascular bypass operations, the diseased artery is shunted using a graft or a healthy vein harvested from the arm or leg.

The purpose of vascular reconstruction is to redirect and augment blood flow or perhaps repair a weakened or aneurysmal vessel through a surgical procedure. The at first glance optimal procedure may not always be best suited, for example, in a patient with complicated or multi-level disease. Pre-operative surgical planning will allow evaluation of different procedures *a priori*, under various physiological states such as rest and exercise, thereby increasing the chances of a positive outcome for the patient.

[1]Some of the material in this section is taken from [2] by Belleman et al. with permission from R.G. Belleman.

4.4.2 Methods used

The aim of the test case presented here is to provide a surgeon with an interactive environment in which vascular reconstruction procedures can be simulated. Such an environment requires the following:

- The environment should be able to present the surgeon with patient specific data at sufficient fidelity so that the patient's infliction can be located. To obtain best understanding on the nature of the problem, the surgeon is presented with a 3D rendering of data obtained from medical scanners (such as CT or MRI) using unambiguous visualization methods. Note that *visual* realism is not the primary goal here; what is more important here is *physical* realism, and then only of particular issues in fluid flow, as discussed later.

- The environment should allow the surgeon to plan a surgical procedure. As the patient's data is presented in 3D, the environment should also allow the surgeon to interact with this data in 3D, offering methods that allow any surgical procedure to be simulated.

- The environment must be able to show the effect of a planned surgical procedure. As the aim of the procedure is to improve the blood flow to the affected area, the surgeon must have some means to compare the flow of blood before and after the planned procedure. A simulation environment is used to accomplish this that calculates *pressure*, *velocity* and *shear stress* of blood flowing through the artery. The visualization environment presents the results of the simulations while the exploration environment allows inspection and probing (qualitatively and quantitatively) of the simulation results (i.e. means should be provided to perform measurements, annotate observations, inspect the flow of the blood stream, etc.).

Lattice Boltzmann simulation

The simulation environment is based on the lattice Boltzmann method (LBM), which is a mesoscopic approach based on the kinetic lattice Boltzmann equation (LBE) for simulating fluid flow [7]. In this method fluid is modelled by particles moving on a regular lattice. At each time step particles propagate to neighboring lattice points and re-distribute their velocities in a local collision phase. This inherent spatial and temporal locality of the update rules makes this method ideal for parallel computing. The LBM allows the calculation of pressure, velocity and shear stress of a fluid flowing through an object.

A grid can only be fed to the LBM if every element is classified to one of four categories:

particle the actual entity performing collisions in the LBE.

inlet/outlet points that define particle entry and exit locations.

boundary all stationary points, impenetrable by any particle.

It is imperative that the classification of all points in the LB grid are consistent, because a gap in the grid can make the simulation behave unexpectedly. Noise or other artifacts that

might have been introduced by the scanning technique can also disrupt simulation behavior and must therefore also be removed. The virtual operating theater provides a way to manually correct these grid inconsistencies where automatic techniques (such as thresholding or clipping) fail.

Before the LBM can be used for vascular simulation the classification of grid elements must be adapted to the vascular system. Fortunately, this is not too difficult, because LB grids bare close resemblance to grids acquired by medical scanners. Since the scan will almost always contains only a subset of the whole (closed) vascular system, the inlet/outlet points define the entry and exit points of the blood, which are represented by particles. The boundary points will generally represent the artery walls.

Visualization of geometry, pressure, velocity, shear stress

The raw MRA data can become very large and also contain non vascular tissue, which makes processing over all the information very inefficient. To retain smooth interaction in the VE, it is desirable to only visualize the artery of interest. This is done in a segmentation phase, which filters out all of the unimportant data [2]. Segmentation results in a 3D grid, and can be done quickly when the patient is injected with contrast fluid while scanning.

Visualization of geometry is done using the hybrid approach as described in section 4.2.4, which makes it possible to achieve smooth interaction using contours and visualization of individual grid points using voxels. The visualization of simulation results (velocity, pressure and shear stress) is not trivial. Specific care must be taken that the simulation results are clearly and nonambiguously presented to the user such that the information that is most important is clearly visible. One of many possible presentation methods is to use hue coloring, although it will be very difficult to visualize all simulation results together. Separate research at the University of Amsterdam is currently specifically focussed on visualization of fluid flow and will not be discussed in this thesis.

4.4.3 Simulating a vascular reconstruction

To test the performance and ease of use of the virtual vascular operating theater, a vascular bypass operation in simulated. In real life surgery one also has to take the position of other internal organs into account. Although there is a high degree of flexibility within a patient's body, it is impossible to, for example, create a bypass through another organ. Currently, such consistency checking is not supported by the virtual operating theater other than through visual inspection. There are roughly two methods with which the task of "drawing" a bypass can be accomplished: individual point editing and implicit modelling.

Grid editing

The most basic level of reconstruction is performed by directly modifying individual elements of the structured grid, which are visualized as voxels. The interaction techniques available to the virtual operating theater include the ability to edit in 3D space (see section 4.2.4). Changing the color of the edit tool changes what kind of LBM element the edited voxels belong to (i.e. inlet, outlet, boundary or fluid point). The color can be transparent

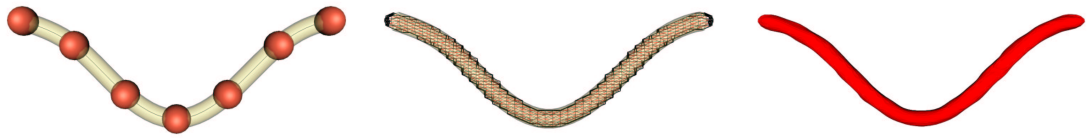


Figure 4.8: Example of how implicit modelling can be used to generate structured grids from polygon input data. Left: A spline defined by 7 anchors. A tube wrapped around the spline defines the diameter of the grid that is to be modelled around the spline. Middle: The spline sampled to a structured grid (in this case 50 x 50 x 50 voxels) using an implicit modeler with the spline as input. The diameter of the tube is used as a maximum distance measure from the spline. Right: Contour representation of the sampled grid.

to remove voxels from the grid which will then be ignored by the LBM. Similar to regular desktop paint programs, the size of the edit tool can be increased to be able to draw a larger volume of voxels at a time. Edited grids can be loaded and saved for further experimentation without.

Inconsistencies in the grid, due to modifications made by the user, can be circumvented with several rules. First of all, it is realistic to allow an editing state in which the grid can not be used in the LBM. In this state, inconsistencies can be automatically determined and directed to the user's attention (e.g. by creating a red haze around problematic voxels). The problem that could arise is that a particle is not enclosed within inlet, outlet and boundary points.

Implicit modelling

Although very useful for fine-tuning, individual voxel modification is not very suitable for large scale reconstruction. Besides increasing the amount of voxels that are modified by increasing pencil size, it is also very useful to be able to draw a line or spline to, for example, simulate a graft bypass or harvested vein. This will make it possible to "draw" e.g. a hollow tube of a certain diameter over the position of the line or spline, greatly simplifying the task of placing a bypass.

The solution to this problem is to include the ability to draw a line between two specified points. One can take line drawing one step further and allow drawing of spline curves defined by several control points (or anchors). Several additional functionality can then be placed on top of these line drawing techniques. Especially for vascular reconstruction the ability to quickly "draw" an artery is very helpful. These hollow tubes, that e.g. represent a bypass, can be generated with the line or spline as their foundation. Generation of splines, the conversion of these splines to tubes of a certain diameter and subsequently the conversion to a structured grid are supported by VTK (see figure 4.8 for an example).

Line and spline drawing is currently not explicitly supported with SCAVI, but has been added with little effort by others using SCAVI objects and events.

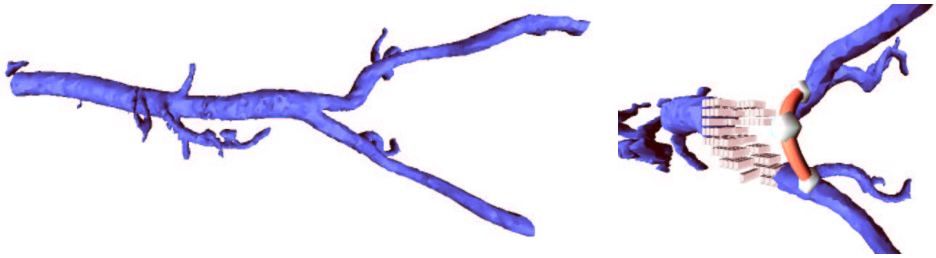


Figure 4.9: Example of drawing a bypass in the virtual vascular operating theater. The left picture depicts the complete "artery" dataset as a contour without sub-sampling. The right picture shows a region represented by voxels sub-sampled by factor 4 and the placement of a spline.

4.4.4 Results

Besides choosing the correct visualization method that clearly lifts out the patient's affliction, this visualization must also be done at sufficient speed to make interaction possible. The visualization accuracy of the abdominal artery dataset compared to the update rate of the VR system is depicted in figure 4.6. The results from section 4.3.2 also apply to this test case.

The grid editing tools based on individual voxels do not provide the necessary tools needed to simulate placement of a bypass. It is very difficult to quickly create a closed tube by editing individual voxels. This is further hampered by the fact that editing only occurs on those instances when the frame is updating. This will create gaps in an otherwise smooth line. An implicit modeler overcomes these problems making the accurate placement of a spline in the operating theater possible within a few minutes (see figure 4.9).

4.5 Conclusions and discussion

The SCAVI toolkit presented in this chapter makes several interaction techniques available to a CAVE developer. Using SCAVI, the developer can focus solely on the task at hand, without the need to create specific manipulation techniques from scratch. The addition of a virtual menu provides the VE with an easy way to incorporate a lot of additional commands, where the number of buttons on a manipulator are too few. The speech recognition system included in SCAVI from chapter 3 proves very useful and has the extra benefit that it is not hampered by low update rates making it a helpful addition next to visual interaction methods. Virtual menus and selection through ray-casting are very intuitive and easy to use, because they are closely related to the familiar WIMP metaphors.

The C++ classes of SCAVI are closely modelled after regular Vtk classes making them very easy to use when already accustomed to Vtk. The SCAVI objects can be used to create more advanced interaction techniques. Indeed, several other developers of VR applications have used SCAVI to implement functions allowing for the interactive placement of simulated vascular bypasses in arterial scans, methods to interactively visualize flow and vector fields

and virtual keyboards for alphanumeric input from within a VE. By using Vtk, it is also not difficult for a developer to use the vast number of available Vtk functionality for complex filtering and analysis, which is refined and growing every day. However, using Vtk does not result in the fastest implementation of interaction methods, mainly because a lot of data must reside in shared memory.

Experimentation has shown that reasonable update rates can be achieved by slightly decreasing the complexity of the object to be visualized. However, removing points from a dataset by sub-sampling can introduce unwanted topology changes. The time to recalculate a contour increases exponentially with the number of triangles makes it unfavorable to edit large grids without sub-sampling or cropping.

Considering the alternative (i.e. manual verification and editing of numerous slices and trying to form a mental picture of all the slices combined) an immersive environment provides a way to remove large areas relatively quickly while retaining a complete overview of the entire dataset. The 3D nature of the VE proves to be extremely beneficial in positioning of the editing cursor as compared to 2D grid editing, which will always require simultaneous viewing of several angles (e.g. top, left and front) as is the case with regular desktop CAD (Computer Aided Design) packages. For example, the table could be removed from the coral dataset in the first test case without enabling voxels making the interaction update rate very high. The standard editing tool is sufficient to make large-scale and delicate modifications to a structured points grid.

The second test case, i.e. the virtual vascular reconstruction operating theater, described in this chapter does not suffice with a standard editing tool. Drawing a closed cylinder, representing a graft bypass or harvested vein in the operating theater, is very difficult to perform by individual voxel editing. The experiment has shown that this placement of a bypass is greatly simplified by using an implicit modeler.

Chapter 5

Conclusions

This thesis started by defining and describing the issues involved in creating an Immersive Virtual Exploration Environment. Interaction with virtual objects in a virtual environment is performed using input devices, which are often combined with tracking technology, and sensory rendering such as vision and audio. Navigational and wayfinding techniques are required to fully explore a virtual world that is larger than the physical space available to the user. The CAVE Automated Virtual Environment was presented as an example of an Immersive Virtual Environment, which uses a wand as input device and a projection based display for visual rendering.

Interaction with virtual objects starts with the identification (or selection) of that object. As with all interaction techniques it is recommended to find WIMP (Windows, Icons, Mouse and Pointer) alternatives for VEs, because they are intuitive and familiar to both novice and expert users. For example, the ray-casting selection method is a direct analogy to the desktop "point-and-click" metaphor. Once an object is selected, actual manipulation can be performed. Efficient and reliable manipulation, such as rotation and translation, can only be accomplished by using constraints which might seem contradictory. One limits the expressiveness of the user to increase the control over the manipulation.

The third chapter describes the different types and characteristics of Automatic Speech Recognition (ASR) systems. Speech recognition provides an intuitive mode of interaction making it possible to give instructions while update rates of the VE are low due to large, computationally expensive virtual scenes. Speech recognition accuracy is hampered by e.g. the type of recognition system, speaking dialect, background noise and the complete amount of freedom of speech the user has. Application context can aid recognition accuracy by minimalizing the vocabulary. A context sensitive speech recognition system (CAVETalk) capable of dynamically activating and deactivating vocabularies was described and implemented, such that only those words in a vocabulary required in a specific context are activated. The ViaVoice speech recognition engine is accurate for American English speakers. However, experimentation has shown that Dutch users speaking English introduce recognition errors due to pronunciation problems. This problem combined with the large global grammar resulted in no difference between context sensitive and context insensitive speech recognition. Especially for novice users, speech selection of objects is typically performed faster compared to menu

selection.

The interaction techniques described in chapter 2 are implemented in the SCAVI multimodal toolkit presented in chapter 4. SCAVI was designed to equip the Virtual Radiology Explorer (VRE) under development at the University of Amsterdam with interaction capabilities. The software packages used in VRE (i.e. VTK and CAVELib) and the target VR system for the SCAVI toolkit are consequently chosen such that inclusion into the VRE could be done relatively easy. SCAVI is based on objects on which manipulation can be performed. The main interaction tool used is ray-casting with the wand. Virtual menus can be created and structured point grids can be modified by an intuitive method of editing. The CAVETalk context sensitive speech recognition system is included in SCAVI, which also implicitly enables a virtual menu with speech.

The efficiency and usability of the interaction techniques developed were validated in two test cases. The test case, i.e. removal of artifacts from a CT scan, showed that the simple editing tool is sufficient for large-scale and delicate modifications to a structured points grid. Visualization of the complete dataset is generally too computationally intensive and will result in a very low update frequency. Cropping or sub-sampling can be used to decrease the complexity of the dataset (i.e. decrease the number of triangles to be visualized) and therefore increase the update frequency. Sub-sampling however introduces unwanted topology changes.

The second test case validated the interaction techniques implemented in SCAVI in a more demanding virtual vascular operating theater, which can be used to validate, plan or rehearse certain surgical procedures in an intuitive way. Experimentation has shown that the task of drawing a bypass is very difficult to perform with the available editing technique. Adding the possibility to "draw" a spline tremendously simplified this task.

5.1 Future work

The SCAVI library currently has a strong disposition towards the CAVE and the wand as the input device. An interaction toolkit can only be successful if it is general enough to be applicable to almost every VR system and a lot of input devices. This would mean that the current mapping of wand buttons to system actions should be made user definable instead of being fixed.

The current implementation of a 2D menu does not allow alignment or glueing on a wall. This can make it difficult to locate the menu when actively navigating the virtual environment. Furthermore, changing the fixed size font used in the menu to a proportional font will make the menu items easier to read when farther away.

The current set of editing tools is not very extensive. A researcher needs to have a wide variety of different tools available to efficiently accomplish an editing task. For example, a selection should be made between the current cubical and a spherical edit tool. Furthermore, a line drawing tool, copy, pasting and deleting a selection are all very helpful to have instantly available in SCAVI.

Editing on the outside of the visible object is straightforward. However, there is currently no method that allows easy editing within the object, since other parts are obscuring the view. Therefore, one must also be able to select which part of the volume is visible. The event based

architecture of SCAVI provides an external solution to this problem by defining eight SCAVI objects that define the visible volume.

Computational steering, calculation of blood flow, shear stress and pressure can only be done if the LB simulation is integrated in the environment. Currently, there is no support for computational steering, grid consistency checking and visualization of quantitative information. It is currently only possible to mark voxels as particular LB elements. The overall speed of the system can be increased by optimizing various algorithms. Another limiting factor is that geometrical output of Vtk actors must be copied into shared memory.

Appendix A

Speech Recognition Control Language

The context sensitive speech recognition (CSSR) experiment of section 3.3 relies on several grammar files that define the vocabulary used for recognition[1]. This chapter gives a short introduction to the format, followed by the contents, of these grammar files.

A.1 Introduction

One way to represent a speech grammar is to use Backus-Naur Form, or BNF. This form of grammar representation is used to describe the syntax of a given language and its notation. These are generally understood and used throughout the world of formal language theory and processing.

This section describes the syntax of a particular type of BNF grammar that has been adapted to the task of speech recognition. It is called Speech Recognition Control Language (abbreviated SRCL and pronounced "circle"). SRCL was developed jointly between the SRAPI (Speech Recognition API) Committee and ECTF (Enterprise Computer Telephony Forum).

Grammars constructed using SRCL offer an organized view of the words and phrases that are part of the speech grammar, since they define a notation for identifying common phrases, optional phrases, and repeated phrases. A speech grammar is defined by enumerating the valid words and phrases. The general form of a SRCL grammar is as follows:

```
<rule> = sentences and phrases.
```

This general form is called a production rule. Every SRCL grammar must start with and contain at least one production rule. The production rule has four parts:

- The left side (`<rule>`): This part of the production rule is required, although it is not necessary to specify the string exactly as we have done. In general, any string may be

[1] The words vocabulary and grammar are used promiscuously in this thesis.

used as long as it is placed between angle brackets <>, and the string formed by the angle brackets and the enclosed word is unique (that is, it is not used elsewhere in the BNF).

- The assignment operator (=): This part of the production rule is required and must be written as shown.
- The right side (sentences and phrases): This part of the production rule defines all of the sentences and phrases that are valid in the speech grammar.
- The end-of-production delimiter (period): This part of the production rule is required and must be specified as shown. Although it is not necessary to separate the period from the sentences-and-phrases section of the rule, doing so often improves readability.

The vertical bar (|) is used to indicate mutually exclusive choices. When the ? operator appears it causes the symbol to its immediate left to be defined as optional. When the + operator appears, it causes the symbol to its immediate left to be defined as one or more of. The * operator is analogous to the + operator, except that it defines zero or more of the symbols to its left.

A colon (:) is used to define annotations that reduce the complexity of parsing command grammar sentences. Key words and phrases are allowed to be marked or annotated when the grammar is defined. Then, when the grammar is used in an application, and whenever an annotated word is recognized, both the word and the annotation are returned to the application. By choosing the annotations properly, most parsing of command grammar sentences can be simplified.

A.2 Grammar's used in the CSSR experiment

A.2.1 Object grammar

This grammar contains 4 words, each of which has probability 0.25 of being recognized.

```
<<start>>
  = "sphere":"SPHERE"
  | "cube":"CUBE"
  | "cone":"CONE"
  | "cylinder":"CYLINDER"
  .
```

A.2.2 Color grammar

This grammar contains 4 words, each of which has probability 0.25 of being recognized.

```
<<start>>
  = "RED":"RED"
  | "GREEN":"GREEN"
  | "YELLOW":"YELLOW"
  | "BLUE":"BLUE"
  .
```

probability	*sentences*	*probability*	*sentences*	*probability*	*sentences*
0.018	33	0.0000017	6380	0.000000016	126720
0.0036	5	0.0000016	4640	0.000000015	38016
0.003	6	0.0000011	25520	0.0000000088	1728
0.006	3	0.00000097	1160	0.0000000055	9280
0.009	8	0.00000086	5760	0.000000005	63360
0.0018	18	0.00000056	9504	0.0000000049	11520
0.00067	9	0.00000032	1728	0.0000000038	204160
0.00043	8	0.0000002	9280	0.0000000036	15312
0.0002	18	0.00000019	15840	0.0000000033	253440
0.00011	174	0.00000018	11520	0.0000000029	2880
0.000062	1160	0.00000014	51040	0.0000000012	25520
0.000054	16	0.00000013	78672	0.00000000081	102080
0.000037	290	0.00000011	2880	0.00000000062	23040
0.000012	432	0.000000079	696	0.00000000042	506880
0.0000077	2320	0.000000045	25520	0.0000000004	38016
0.0000069	2880	0.000000044	4640	0.00000000013	63360
0.000005	3828	0.00000003	102080	0.0000000001	204160
0.0000041	720	0.000000026	1160	0.000000000091	253440
0.0000029	696	0.000000023	23040	0.000000000011	506880

Table A.1: Overview of the number of sentences in the global grammar with a particular recognition probability.

A.2.3 Rank grammar

This grammar contains 4 words, each of which has probability 0.25 of being recognized.

```
<<start>>
  = "first":"ONE"
  | "second":"TWO"
  | "third":"THREE"
  | "fourth":"FOUR"
  .
```

A.2.4 Global grammar

The global grammar contains 130 words, which can be combined to form 2.843.476 distinct sentences. Not all of these sentences have equal recognition probability. Only a small group of words contribute to a large part of the total recognition probability (see table A.1).

```
<<start>>
  = "computer":"COMPUTER"
  | "create":"CREATE" <object>
  | "paste":"PASTE"
  | "select":"SELECT" <object>
  | "deselect":"DESELECT"
  | "unselect":"DESELECT"
  | <quit>
```

```
  | <help>
  | <again>
  | <menu> "menu":"MENU"
  | <move>? <direction>
  | <move>? <direction> <number> <point>? <meters>?
  | "select":"SELECT"
  | "set"? <scale> <number> <point>? "percent"?
  | "rotate":"ROTATE" <axis> <number> <point>? "degrees"?
  | "rotate":"ROTATE X" <number> <point>? "degrees"?
  | "copy":"COPY"
  | <delete>
  | <bigger>
  | <smaller>
  | <setcolor> <color>
  | "show object":"SHOW OBJECT"
  | "hide object":"HIDE OBJECT"
  .

<delete>
  = "cut":"DELETE"
  | "delete":"DELETE"
  | "remove":"DELETE"
  | "erase":"DELETE"
  | "destroy":"DELETE"
  | "kill":"DELETE"
  .

<setcolor>
  = "make":"COLOR" "color"?
  | "set" "color":"COLOR"
  | "paint":"COLOR" "color"?
  .

<color>
  = "white" : "1.0 1.0 1.0"
  | "black" : "0.0 0.0 0.0"
  | "green" : "0.0 1.0 0.0"
  | "blue"  : "0.0 0.0 1.0"
  | "red"   : "1.0 0.0 0.0"
  | "purple": "1.0 0.0 1.0"
  | "yellow": "1.0 1.0 0.0"
  | "orange": "1.0 0.5 0.0"
  | "cyan"  : "0.0 1.0 1.0"
  .

<point>
  = "point":"." <number-1>
  | "point":"." "oh":"0"
  | "point":"." "zero":"0"
```

```
  | "dot":"." <number-1>
  | "dot":"." "oh":"0"
  | "dot":"." "zero":"0"
  .

<axis>
  = "x":"X"
  | "y":"Y"
  | "z":"Z"
  .

<meters>
  = "meters"
  | "feet"
  | "foot"
  | "units"
  .

<move>
  = "move"
  | "translate"
  .

<scale>
  = "scale":"SCALE"
  | "size":"SCALE"
  .

<number-1>
  = one:"1"
  | two:"2"
  | three:"3"
  | four:"4"
  | five:"5"
  | six:"6"
  | seven:"7"
  | eight:"8"
  | nine:"9"
  .

<number-2>
  = twenty:"2"
  | thirty:"3"
  | fourty:"4"
  | fifty:"5"
  | sixty:"6"
  | seventy:"7"
  | eighty:"8"
  | ninety:"9"
```

```
  .

<number-99>
  = oh:"00"
  | zero:"00"
  | one:"01"
  | two:"02"
  | three:"03"
  | four:"04"
  | five:"05"
  | six:"06"
  | seven:"07"
  | eight:"08"
  | nine:"09"
  | ten:"10"
  | eleven:"11"
  | twelve:"12"
  | thirteen:"13"
  | fourteen:"14"
  | fifteen:"15"
  | sixteen:"16"
  | seventeen:"17"
  | eighteen:"18"
  | nineteen:"19"
  | twenty:"20"
  | thirty:"30"
  | fourty:"40"
  | fifty:"50"
  | sixty:"60"
  | seventy:"70"
  | eighty:"80"
  | ninety:"90"
  | <number-2> <number-1>
  .

<number-3>
  = "one"? "hundred":"1" "and"?
  | "two" "hundred":"2" "and"?
  | "three" "hundred":"3" "and"?
  .

<number>
  = <number-99>
  | <number-3> <number-99>
  .

<direction>
  = "left":"LEFT"
  | "right":"RIGHT"
```

```
  | "down":"DOWN"
  | "up":"UP"
  | "front":"FRONT"
  | "forward":"FRONT"
  | "back":"BACK"
  | "backward":"BACK"
  .

<smaller>
  = "make"? "smaller":"SMALLER"
  | "shrink":"SMALLER"
  | "scale down":"SMALLER"
  .

<bigger>
  = "make"? "bigger":"LARGER"
  | "enlarge":"LARGER"
  | "make"? "larger":"LARGER"
  | "scale up":"LARGER"
  .

<menu>
  = "open":"OPEN"
  | "spawn":"OPEN"
  | "show":"OPEN"
  | "view":"OPEN"
  | "hide":"CLOSE"
  | "close":"CLOSE"
  | "exit":"CLOSE"
  | "leave":"CLOSE"
  .

<again>
  = "again":"AGAIN"
  | "repeat":"AGAIN"
  .

<object>
  = "sphere":"SPHERE"
  | "cube":"CUBE"
  | "box":"CUBE"
  | "cone":"CONE"
  | "cylinder":"CYLINDER"
  .

<quit>
  = "quit":"QUIT" <program>
  | "exit":"QUIT" <program>
  .
```

```
<program>
  = "program"
  | "application"
  .

<help>
  = "help":"HELP"
  | "help me":"HELP"
  | "give" "me"? "assistance":"HELP"
  | "provide" "me"? "with" "assistance":"HELP"
  .
```

Bibliography

[1] R.G. Belleman, J.A. Kaandorp, D. Dijkman, and P.M.A. Sloot. GEOPROVE: Geometric probes for virtual environments. In P.M.A. Sloot, M. Bubak, A. Hoekstra, and L.O. Hertzberger, editors, *High Performance Computing and Networking (HPCN'99)*, pages 817–827, Amsterdam, the Netherlands, April 1999. Springer-Verlag.

[2] R.G. Belleman and P.M.A. Sloot. The design of dynamic exploration environments for computational steering simulations. In Marian Bubak, Jacek Mościński, and Marian Noga, editors, *Proceedings of the SGI Users' Conference*, pages 57–74, Kraków, Poland, October 2000. Academic Computer Centre CYFRONET AGH.

[3] R.G. Belleman, B. Stolk, and R. de Vries. Immersive virtual reality on commodity hardware. In R.L. Lagendijk, J.W.J. Heijnsdijk, A.D. Pimentel, and M.H.F. Wilkinson, editors, *Proceedings of the seventh annual conference of the Advanced School for Computing and Imaging*, pages 297–304, Heijen, the Netherlands, May 30-June 1 2001. Advanced School for Computing and Imaging (ASCI).

[4] Doug A. Bowman. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *Symposium on Interactive 3D Graphics*, 1997.

[5] Doug A. Bowman and Larry F. Hodges. User interface constraints for immersive virtual environment applications. In *Proceedings of IEEE VRAIS*, pages 35–38, 1997.

[6] J. Butterworth, A. Davidson, S. Hench, and T.M. Olano. 3DM: A three-dimensional modeler using a head-mounted display. In *ACM Computer Graphics: Proceedings of 1992 Symposium on Interactive 3D Graphics*, pages 135–138, Cambridge, MA, 1992.

[7] S. Chen and G.D. Doolen. Lattice Boltzmann method for fluid flows. *Annu. Rev. Fluid Mech.*, 30:329, 1998.

[8] CommWeb.com. Automated speech recognition, September 14, 2000. On the web: `http://www.commweb.com/article/COM20000914S0008`.

[9] C. Cruz-Neira, D.J. Sandin, and T.A. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In *SIGGRAPH '93 Computer Graphics Conference*, pages 135–142. ACM SIGGRAPH, August 1993.

[10] Rudolph P. Darken. Wayfinding in large-scale virtual worlds. In *CHI '95 Conference Proceedings*, pages 45–46, May 7-1, 1995.

[11] Rudy Darken. Hands-off interaction with menus in virtual spaces. In *Proceedings of SPIE 1994*, volume 2177, pages 365–371, 1994.

[12] T. Todd Elvins, David R. Nadeau, and David Kirsh. Worldlets - 3D thumbnails for wayfinding in virtual environments. In *UIST 97*, pages 21–30, 1997.

[13] Matthew Hall. *VTK to CAVE Translator library*, October 18, 1999. `http://zeus.ncsa.uiuc.edu/~mahall/`.

[14] IBM Corporation, Armonk, NY. *Data Explorer Reference Manual*, 1991.

[15] International Business Machines (IBM) Corp. *IBM ViaVoice Software Development's Kit*, 2000. `http://www.ibm.com/software/speech/dev/`.

[16] Christian Knöpfle. Interfacing with simulation data in an immersive environment. In *Proceedings of the sixth Eurographics Workshop on Virtual Environments*, pages 157–166, Amsterdam, June 1-2, 2000.

[17] Evin Levey, Christopher Peters, and Carol O'Sullivan. New metrics for evaluation of collision detection techniques. In *Proceedings of The 8-th International Conference in Central Europe on Computer Graphics Visualization and Interactive Digital Media'2000, Plzen*, 2000.

[18] Robert W. Lindeman, John L. Sibert, and James K. Hahn. Towards usable VR: An empirical study of user interfaces for immersive virtual environments. In *CHI '99 Conference Proceedings*, pages 64–71, May 15-20, 1999.

[19] I. Scott MacKenzie and Colin Ware. Lag as a determinant of human performance in interactive systems. In *INTERCHI'93 Conference on Human Factors in Computing Systems*, pages 488–493, Amsterdam, April 24-29, 1993.

[20] Mark R. Mine. ISAAC: A virtual environment tool for the interactive construction of virtual worlds. Technical report, University of North Carolina, 1995. Computer Science Technical Report TR95-020.

[21] Mark R. Mine. Virtual environment interaction techniques. Technical report, University of North Carolina, 1995. Computer Science Technical Report TR95-018.

[22] Mark R. Mine. Working in a virtual world: Interaction techniques used in the chapel hill immersive modeling program. Technical report, University of North Carolina, 1996. Computer Science Technical Report TR96-029.

[23] Mark R. Mine. Moving cows in space: Exploiting proprioception as a framework for virtual environment interactions. Technical report, University of North Carolina, 1997. Computer Science Technical Report TR97-003.

[24] Mark R. Mine, Frederick P. Brooks Jr., and Carlo H. Sequin. Moving objects in space: Exploiting proprioception in virtual-environment interaction. In *Proceedings of SIGGRAPH 1997*, 1997.

[25] Jurriaan D. Mulder. Remote object translation methods for immersive virtual environments. In *Proceedings of Eurographics Virtual Environments '98*. Springer, 1998.

[26] The Numerical Algorithms Group Ltd., Oxford, UK. *Iris Explorer User's Guide*, 1998. `http://www.nag.co.uk/visual/IE/iecbb/DOC/UG/CONTENTS.html`.

[27] Sharon Oviatt. Taming recognition errors with a multimodal interface. *Communications of the ACM*, 43(9):45–52, September 2000.

[28] Steven G. Parker and Christopher R. Johnson. SCIRun: A scientific programming environment for computational steering. In *Supercomputing '95*, 1995.

[29] Ivan Poupyrev, Suzanne Weghorst, Mark Billighurst, and Tadao Ichikawa. A framework and testbed for studying manipulation techniques for immersive VR. In *Proceedings of ACM Conference VRST*, pages 21–28, 1997.

[30] Silicon Graphics Inc. Software Products. Performer homepage, 2001. On the web: `http://www.sgi.com/software/performer/`.

[31] W. Robinett and R. Holloway. Implementation of flying, scaling and grabbing in virtual worlds. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, pages 197–208, 1992.

[32] Michael Rorke, Shaun Bangay, and Peter Wentworth. Virtual reality interaction techniques. In *Proceedings of the 1st Annual South African Telecommunication, Networks and Application Conference (SATNAC)*, pages 526–532, Cape Town, South Africa, September, 1998. On the web: `http://cs.ru.ac.za/homes/g97rc001/ref.html`.

[33] SARA Computing and Networking Services, Amsterdam, the Netherlands. *SARA Homepage*, 2001. `http://www.sara.nl/`.

[34] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit, an object-oriented approach to 3D graphics (2nd edition)*. Prentice Hall, Upper Saddle River, NJ, 1997. ISBN 0-13-954694-4.

[35] Loki Entertainment Software. OpenAL homepage, 2000. On the web: `http://www.openal.com`.

[36] R. Stoakley, M.J. Conway, and R. Pausch. Virtual reality on a WIM: Interactive worlds in miniature. In *Proceedings of SIGCHI 1995*, 1995.

[37] Helmer Strik and Catia Cuchiarini. Modeling pronunciation variation for ASR: Overview and comparison of methods. Technical report, Dept. of Language and Speech, University of Nijmegen, 1998.

[38] Gnanasekaran Swaminathan. *C++ Socket Classes library*, January 1994. `ftp://ftp.virginia.edu/pub/tools/`.

[39] Tapio Takala and James Hahn. Sound rendering. *SIGGRAPH '92*, 2:211–220, July 26-31, 1992.

[40] Markku Turunen. Error handling in speech user interfaces in the context of virtual worlds. In *Proceedings of ACHCI'98*, pages 68–75, Tampere, Finland, 1998.

[41] Robert van Liere, Jurriaan D. Mulder, and Jarke J. van Wijk. Computational steering. In H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, editors, *High-Performance Computing and Networking*, pages 696–702. Springer-Verlag, April 1996.

[42] Dan Venoilia. Facile 3D direct manipulation. In *INTERCHI'93 Conference on Human Factors in Computing Systems*, pages 31–36, Amsterdam, April 24-29, 1993.

[43] Virtual Reality Consulting Inc., Chicago, IL. *CAVE User's Guide*, 1998. `http://www.vrco.com/CAVE_USER/`.

[44] Matthias M. Wloka and Eliot Greenfield. The virtual tricorder: A uniform interface for virtual reality. In *UIST '95: The Eighth Annual Symposium on User Interface Software and Technology*, Pittsburgh, PA, 1995.

[45] Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide (second edition)*. Addison-Wesley, 1996. ISBN 0-201-46138-2.

[46] Nicole Yankelovich, Gina-Anne Levow, and Matt Marx. Designing SpeechActs: Issues in speech user interfaces. In *CHI '95 Conference Proceedings*, pages 369–376, May 7-11, 1995.

[47] Klaus Zechner and Alex Waibel. Using chunk based partial parsing of spontaneous speech in unrestricted domains for reducing word error rate in speech recognition. Technical report, Language Technology Institute, Carnegie Mellon University, Pittsburgh, PA, 1997.