ACTS Project N. AC040



D2.6

Guidelines for building CVE applications

Project Number:	ACTS Project N. AC040
Project Title:	COVEN - Collaborative virtual environments
Deliverable Type:	P*

CEC Deliverable Number:	AC040-KPN-RESEARCH-DS-P-026.b1
Contractual Date of Delivery to the CEC:	June 30, 1997
Actual Date of Delivery to the CEC:	August 8th 1997
Title of Deliverable:	Guidelines for building CVE applications
Workpackage contributing to the Deliverable:	WP2
Nature of the Deliverable:	R**
Author(s):	EPFL, Geneva, IIS, TNO, Lancaster, UCL, Nottingham, SICS, Division, Thomson, KPN

Abstract:
As part of the European ACTS programme, COVEN has made guidelines for building a Collaborative Virtual Environments (CVEs), based on the experiences of the COVEN demonstrators. This deliverable provides guidelines for the decisionmaker, the designer, the developer and the modeler in order to build Collaborative Virtual Environment applications.

Keyword list:
Collaborative Virtual Environments, Guidelines, Usability, VE, CVE Design

***Type: P-public, R-restricted, L-limited, I-internal**
****Nature: P-Prototype, R-Report, S-Specification, T-Tool, O-Other**

Executive Summary

COVEN is focused on the development of a computational service for teleworking and virtual presence. The overall objective of the project is to provide the facilities needed to support future co-operative teleworking systems.

At this moment in the project, first versions of two COVEN demonstrators have been built in order to gain experience with the development of CVE applications and the virtual reality platform dVS (distributed visualisation system). This document is based on these experiences, and on related expertise of the partners in COVEN. Building a successful CVE is still very much an art, and involves various skills from different people. This document describes **guidelines** people will need, when making a CVE for a given task.

A VE-system is a **computer system** which generates **a 3-dimensional virtual environment** with which the user can **interact** in such a way that he receives **real time feedback**. A **Shared VE-system or SVE** is a VE-system that can be used by **multiple, simultaneous users**, which can be considered **part of the virtual environment** themselves. Hence, a user may be able to interact with other users. A **collaborative VE system or CVE** is an SVE aimed at a **collaborative** task.

Building a CVE can be described with a basic **route** from a task to be performed to a CVE implementation for that task. In the description of this route we can identify various **roles**, responsible for realising parts of the route. Each of the roles in the process of making a CVE must answer specific **questions**, in order to produce the results for its particular portion. By making those questions explicit, and the issues and parameters involved, we will know what guidelines we need. I.e., **guidelines** can be considered approximate answers to those very questions.

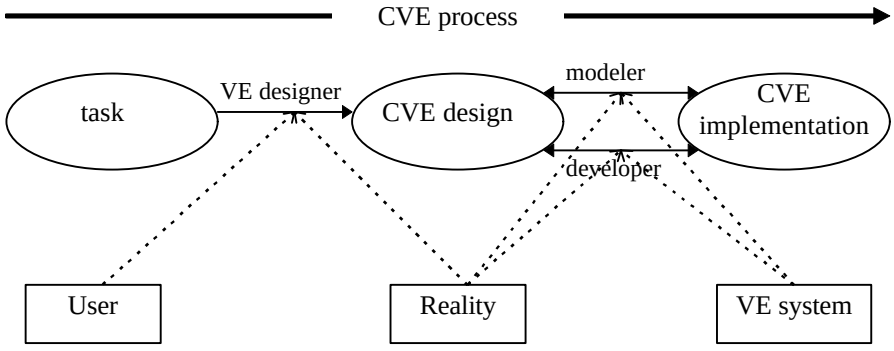


Figure 1 Building process of a CVE

Each of the chapters in this document is aimed at a different role in the CVE building process. First, there is the **decision maker**, who is responsible for the complete process of building a CVE. He needs to answer three main questions when deciding whether or not to build a CVE for a particular task:

1. Should I build a CVE for the particular task I have in mind?
2. What are the costs and benefits of building a CVE for this particular task?
3. What factors can influence the project to become a success or a failure?

There are several criteria that can be used to determine when a task is suitable for implementation in a CVE. Three tasks that seem particularly suited to CVE implementation are 3D product prototyping, training and data visualisation.

Cost and benefit factors can be derived from known CVE and similar high-tech IT projects. These factors can be used as checklist when determining whether the costs of a CVE balance properly against the expected benefits. In addition to this, from these experiences a checklist has been made that can be used when deciding between different information technology projects with a limited budget.
Finally, from earlier projects we can find the success factors, fail factors and risks. Success factors positively influence the acceptance of a CVE by the intended users, fail factors may negatively influence this acceptance, and the risks are inherent to the use of new technology such as CVE technology.

Once it has been decided to build a CVE, the **designer** is asked to translate the given task into a design, i.e., a description of a virtual environment including all objects and their behaviours. The designer has to deal with three basic questions:
1. What does a user see in the VE?
2. How can a user learn from the things he sees in the CVE?
3. How can you make a CVE usable?

To answer the first two questions, the designer should take into account how users would act, react, and learn in a real environment. A virtual environment should be designed to be compatible with this natural behaviour. If the implications of this for a virtual environment are clear the designer can build a good interface, taking into account the “general” rules of Nielsen: an interface which is meant for a specific user for a specific task.

The **modeler** and the **developer** must translate the design into an actual CVE implementation. The modeler is mainly concerned with the visual aspect of the virtual environment, while the developer combines the visual aspect of an object with the desired behaviour for the given task.
Both the modeler and the developer are constantly asking two questions:
- What does the VE-designer try to achieve with this particular object in the VE design? So how should I visualise this object and let it behave?
- How can I avoid the limitations of the system underlying the VE?

For the developer, implementations of tasks can be reduced mostly to good implementations of the nine generic functions for supporting collaborative activities identified in earlier WP2 deliverables:
- mutual awareness,
- communication,
- human actors,
- subjective views,
- resource management,
- object manipulation,
- participants roles and rights,
- group navigation,
- global map.

Human actors and subjective views are probably the most important elements from the developer point of view. Good implementations of these two enable many aspects of the other generic functions to be easy derivatives.

The modeler is focused on providing a virtual environment which looks good and reacts to actions from the user without delay. Users should be able to navigate smoothly and continuously through the virtual world, with the scene rendered as soon as the users or some objects moves. The modeler will concentrate on the actions that cost the most effort (in computer time and memory) of the underlying CVE architecture. These actions are: rendering, collision detection between objects or intersection of rays and objects. The basic techniques to improve these are
1. Simplifying the scene.
2. Dividing the scene into chunks in order to help the application performing optimisations. These techniques include spatial organisation and creation of nested hierarchies of objects.
3. Giving hints to the application about the shapes used. These techniques include using geometric primitives, using LOD (Level of Details) and setting geometric hints.

Contents

1. INTRODUCTION OF GUIDELINES FOR MAKING A CVE 8

1.1 Definition and use of a CVE 8

1.2 Making a CVE 9
1.2.1 Decision Maker 10
1.2.2 VE Designer 10
1.2.3 Developer 10
1.2.4 Modeler 10

2. GUIDELINES FOR THE CVE DECISION MAKER 12

2.1 Tasks in CVEs 12
2.1.1 Prototyping a complex 3D product 13
2.1.2 Training 13
2.1.3 Data visualisation 13

2.2 Cost and benefits of a CVE for a given task 14
2.2.1 Costs of Collaborative Virtual Environments 14
2.2.2 Benefits of Collaborative Virtual Environments 15
2.2.3 Comparing projects 16

2.3 Success and Fail factors of CVEs 16
2.3.1 Success factors 16
2.3.2 Fail factors 17

2.4 Summary 17

3. GUIDELINES FOR THE CVE DESIGNER 19

3.1 What does a user see? 19
3.1.1 Awareness 19
3.1.2 Attention 21
3.1.3 Perception of space 22
3.1.4 Perception of motion 23

3.2 How does a user learn 24
3.2.1 Learning from interaction 24

3.2.2 Learning in a virtual world....25

3.3 How can you make a CVE usable?....26
3.3.1 A usable interface....26
3.3.2 Metaphors....27

3.4 Conclusion....28

3.5 References....28

4. GUIDELINES FOR THE DEVELOPER....29

4.1 Human actors....29
4.1.1 What is needed to represent a virtual human body....30
4.1.2 How to control a virtual human....32
4.1.3 How can virtual humans be applied in terms of the generic functions for CVEs....34
4.1.4 General guidelines for implementation of virtual humans....35

4.2 Subjective views....35
4.2.1 What is needed to represent subjective views....36
4.2.2 How to control subjective views....37
4.2.3 How can subjective views be applied in terms of the generic functions for CVEs....37
4.2.4 General guidelines for implementation of subjective views....37

4.3 Mutual awareness....38
4.3.1 Recognition of identity....38
4.3.2 Role of participants....39
4.3.3 Perception of current activity....39
4.3.4 Awareness depending on distance....40
4.3.5 Awareness depending on task context....40

4.4 Communication....40
4.4.1 Guideline for developer....42

4.5 Resource management....42
4.5.1 Avoiding resource conflicts....42
4.5.2 Use social protocols to solve resource conflicts....43
4.5.3 Protect resources through some mechanism....43
4.5.4 Guidelines for developer....43

4.6 Participants roles and rights....43
4.6.1 Requirements for COVEN demonstrators....44
4.6.2 Possible implementations....44
4.6.3 Current design choice....45
4.6.4 Guideline for developer....45

4.7 Object manipulation....45
4.7.1 Requirements for COVEN demonstrators....46

4.7.2 Implementation of an object manipulation tool....46

4.8 Global map....46
4.8.1 Requirements for COVEN demonstrators....47
4.8.2 A possible implementation....47
4.8.3 Current design choice in the COVEN demonstrators....48
4.8.4 Guidelines for the developer....49

4.9 Group functionality....49
4.9.1 Requirements for COVEN demonstrators....49
4.9.2 Possible implementations....50
4.9.3 Current design choice....50
4.9.4 Guidelines for developer....50

5. GUIDELINES FOR THE CVE MODELER....51

5.1 Simplified architecture of a CVE....51

5.2 Simplifying the scene....52
5.2.1 Reducing polygon count....53
5.2.2 Reducing the number of vertices....54
5.2.3 Using textures....54
5.2.4 Using appropriate textures....55
5.2.5 Reducing the number of lights....55
5.2.6 Using billboards....56
5.2.7 Using anti-aliasing....57

5.3 Dividing the scene....57
5.3.1 Creating hierarchies of objects....57
5.3.2 Arranging objects spatially....57
5.3.3 Using portals....58

5.4 Helping the VR application....58
5.4.1 Using high-level primitives....58
5.4.2 Using Level-of-detail....59
5.4.3 Geometry hints....59

5.5 Conclusion....60

5.6 References....60

List of Figures

Figure 1 Building process of a CVE 2
Figure 2 Building process of a CVE 9
Figure 3 The Conceptual Knowledge Framework, illustrated with the example of a generic post office. 22
Figure 4 The acquisition of knowledge from interaction with the (virtual) world which leads to cognitive immersion. A) Presence. B) Telepresence. (A and B are adapted from: Loomis, (1992)), and C) Immersion (Tromp, 1995). 24
Figure 5 The virtual body skeleton (Joints) 31
Figure 6 A humanoid controlled by a real user in dVS 32
Figure 7 Adding an autonomous crowd in a virtual public place 33
Figure 8 Basic set of gestures and postures 34
Figure 9 (a) Happy Walking (b) Sad Walking 35
Figure 10 A generic view matrix 36
Figure 11 A view-matrix for a male human 36
Figure 12 Two user's (subjective) views of the same shared space 37
Figure 13 The Teleconference Room 42
Figure 14 The 'Drive' interface 46
Figure 15 Example of a global map in DIVE 48
Figure 16 Global map and (subjective) path in virtual environment 48
Figure 17 In this figure, the cow to the left is not drawn with Gouraud shading, while the cow to the right is. 53
Figure 18 Use of texture 55
Figure 19 Examples of two different methods of radiosity 56
Figure 20 Example of trees as billboards 57
Figure 21 A detailed cylinder to the left and a simplified cylinder to the right 59
Figure 22 Example of automatic LOD generation 59

1. Introduction of guidelines for making a CVE

This document describes the different kinds of guidelines we would like to have, when making a CVE for a given task.

First, in order to have a common ground for discussion, we must give some **definition** of what we consider a CVE and why anyone would want to make one. Then we describe the basic **route** from a task to be performed to a CVE implementation for that task. In the description of this route we can identify various **roles**, responsible for parts of the route. For instance, the role of a CVE designer is to translate a given task into a design of a CVE, i.e. a description of a virtual environment and the behaviours of the elements in it.

Each of the roles in the process of making a CVE must answer specific **questions**, in order to produce the results for its particular portion. By making those questions explicit, and the issues and parameters involved, we will know what guidelines we need. I.e., *guidelines can be considered approximate answers to those very questions*.

1.1 Definition and use of a CVE

We can consider a virtual environment or VE-system as a means for a user to perform a given task, or as a medium in which this task can be performed (for our purpose this makes no difference).

As a definition for a **VE-system** we consider the following, which lists a few necessary elements:
- A VE-system is a **computer system**,
- which generates **a 3-dimensional virtual environment,**
- with which the user can **interact**,
- in such a way that he receives **real time feedback**.

A **Shared VE-system or SVE** would extend this definition with the following requirements:
- The VE-system can be used by **multiple, simultaneous users**,
- which can be considered **part of the virtual environment** themselves. (Hence, a user may be able to interact with other users.)

A **Collaborative VE-system or CVE** would then be an extension of an SVE:
- A CVE is an SVE aimed at a **collaborative** task.

We can think of a CVE as a virtual environment or **world**, populated by **objects** and representations of the user(s) called **avatars**. With objects **behaviour** is associated (sounds, movement, interaction with the user or other objects).

These definitions list the necessary requirements. However, there are many aspects in which VE-systems can vary. These aspects can often be considered **dimensions**, on which VE-designers can make choices when designing a VE-system. To name a few aspects:
- **Presence**, the degree to which the user feels “part of” the virtual environment. This degree is dependent on many other aspects of the VE-system, but is probably critical to the success or failure of the task execution by the user.
- **Immersiveness**, the degree to which the virtual environment surrounds the user. Desk top systems, Head mounted displays, CAVE systems are some of the values along this dimension.

- **Interaction with reality**, the degree to which the user must interact with reality while at the same time in the VE-system. For instance, what happens in the reality around him is of no concern to a user playing Quake, whereas for an Enhanced Reality task the virtual environment provides assistance for a task in reality.
- Is the VE-system **distributed** or not? (Are the users at the same location?)
- Does the virtual world have **persistence**? (Is it still there when the user logs off). For instance, for the building task at hand in Alpha World it is desirable that the world has persistence, whereas in Quake this is not (every time the game starts, you start afresh).
- What are the **input/output means** used for the VE-system? (HMD, space mouse, data glove, speech, head tracking?) The input/output means determine which senses of the user are involved directly in the operation of the VE system.

1.2 Making a CVE

Basically, a CVE system is just a regular information technology system. This means that well-established software development methods like SDM can be used to realise a CVE system. Such methods describe what products should be realised at what stage of the project (e.g., a conceptual design, a test plan, an acceptance test, etcetera). However, a general method like SDM does not tell you *how* to realise those products specifically for a CVE. In order to help with this, guidelines might be useful.

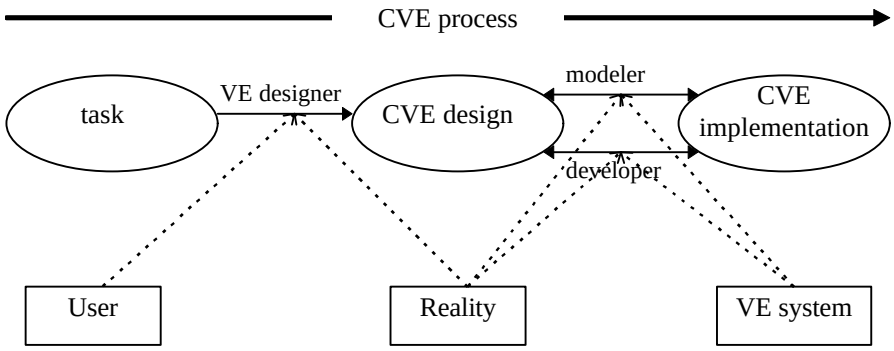


Figure 2 Building process of a CVE

The main steps in the realisation of a CVE system are illustrated above. We assume that a certain task is given, and a VE designer creates a **design** for a CVE system to support the user in executing this task. A CVE design consists of:

- description of the virtual world, and the objects in it,
- description of the behaviour of the objects in the virtual world. “Behaviour” can be taken to mean all non-visual properties of the object, but specifically its behaviour with respect to other objects and the user(s):
    - what happens if the object collides with an other object,
    - what happens when the user tries to select, move or destroy the object.

The design of the CVE is then translated into a CVE implementation for a given system. The visual aspect of the CVE is usually referred to as the model. The behaviour is translated into object functions, and attached to the model. Since the CVE implementation combine into often limited by the possibilities of the VE system, design and implementation is a cyclic process: limits on the system necessitate changes in the design, which in turn necessitate changes in the implementation.

The deliverable 2.6 “Guidelines for making a CVE” addresses the guidelines for the different roles in different chapters.

- Chapter 2: Guidelines for the decision maker
- Chapter 3: Guidelines for the VE designer
- Chapter 4: Guidelines for the developer
- Chapter 5: Guidelines for the modeler

1.2.1 Decision Maker

Decision makers want to judge the entire route from task-description to CVE implementation, in order to decide whether it is worthwhile to invest money in it (usually). They want to know: for this particular task, is it **desirable** to make a CVE system for it? A more concrete interpretation of this question could be:

- what are the **costs and benefits** of building a CVE for this particular task? Since in many cases these costs and benefits can only be established qualitatively, guidelines may help here.

Also, decision makers want to know if it is **feasible** to build a CVE system for a given task. This could determine the degree of risk they are willing to take.
In order to get guidelines for this question, we can look at successes and failures in the past, and try to extract the critical factors from those experiences.

1.2.2 VE Designer

The VE designer translates the task for the user into a design for a CVE system. In doing so, the designer will always focus on the intended users of the CVE system: how will they react to the virtual world put before them in the CVE? Will they understand this world easily enough, in order for them to execute their intended task without unnecessary difficulty? Is the world "natural" enough?

Since most of our experience with users is from tasks in a real environment, or, if it is with computer related tasks, in a 2D graphical environment, it is straightforward to use these experiences when designing 3D objects in a virtual world. For instance, almost any user will recognise a button as something which can be pushed with a certain effect. So if we make a (recognisable) 3D button and label it appropriately, e.g., "open door", most users will automatically assume that pushing that button will open a door (probably close by) in the virtual environment.
However, since a virtual world need not be bound by the same laws of nature as reality, but only by the "laws" of what would be natural to the user, a virtual world allows a much greater freedom to the designer. There is no reason why the user crossing a given threshold couldn't open the door, without the need for a button. Unfortunately, it is not always obvious what would be "natural" for an intended user, hence there is a need for guidelines in the design.

1.2.3 Developer

The task of the developer is to translate the *behaviour* aspects of the CVE design into an actual implementation. The developer is constantly asking two questions:

- What does the VE-designer try to achieve with this particular object in the VE design? So how should I visualise this object, and let it behave?
- How can I avoid the limitations of the system underlying the VE?

The emphasis for the developer is on the behaviour of objects.

For example, for the CD-player in the citizen application, part of the behaviour of such an object in *reality* is that the user must position the CD in the tray, close the tray and push the PLAY button. However, since the goal of the VE-designer is not to accurately simulate a CD-player but merely to provide an object for distributing information, and since a limitation of most present VE systems is that it is hard to manipulate an object in the virtual world, the developer can choose to replace this behaviour by an animated sequence.

1.2.4 Modeler

The task of the modeler is to translate the *visual* aspect of the CVE design into an actual implementation. The questions the modeler asks are the same as for the developer, except that the emphasis is on the appearance of objects instead of on their behaviour.

For example, in the citizen application the VE-designer proposes a CD-player to distribute information about Rhodes. If this CD-player is visualised as a black box, there is a danger that the user will not recognise it as a CD-player and hence not use it. If, on the other hand, the CD-player is overly detailed with several buttons etcetera, this will increase the number of polygons in the VE world and hence possibly slow down the system.

Guidelines for the modeler fall in two basic categories:

- what aspects of the visual appearance of objects in reality must be recreated in the virtual world to achieve the desired effect (functionality)?
- In what way can the limitations of underlying VE systems (software and hardware) be overcome for this purpose?

2. Guidelines for the CVE decision maker

This chapter provides guidelines for the decision maker in a CVE project. Decision makers are responsible for deciding whether or not to build a Collaborative Virtual Environment (CVE) for a particular task. Ultimately, they will decide whether it is worthwhile to invest in a CVE project. For this, they need information about the entire route from task description to CVE implementation.

The three main questions a decision maker needs to answer are:
1. Should I build a CVE for the particular task I have in mind?
2. What are the costs and benefits of building a CVE for this particular task?
3. What factors can influence the project to become a success or a failure?

The next three paragraphs will provide guidelines that help the decision maker to formulate answers to these three main questions. The guidelines are accompanied by examples and references.

2.1 Tasks in CVEs

A decision maker should decide whether a Collaborative Virtual Environment is a useful implementation for a given task. To do this, the decision maker needs general requirements. From literature and from our own experience, we gather that VEs in general should be used with tasks that:
1. Are difficult to realise
2. Require 3D interactive graphics
3. Require immersion

Situations that are difficult to realise in real-life can benefit from the simulation environment of a VE. Examples of such situations include:
- Tasks that are expensive to perform in the real world, for example, fighter pilot training is expensive in fuel costs or property destruction.
- Situations that are too dangerous or unsafe, for example when injury or loss of life might be incurred during more realistic exercises.
- Situations with difficult logistics, for example when large distances would have to be travelled by people that need to collaborate.
- To show things that haven't been produced yet, for example in product prototyping or consumer tests.

Situations that require 3D interactive graphics are situations that benefit from:
- Direct feedback
- Natural interaction and communication
- Addition of a 3rd dimension

Examples of such situations include:
- Idea development
- Designing complex 3D prototypes
- Gaining insight into complex technical drawings
- Pattern recognition

Situations that require immersion are situations where involvement or concentration is required. Examples of such situations include:
- Training
- Salesperson-client interactions
- Games

In addition to these general VE requirements, **Collaborative** Virtual Environments are needed when a task needs to be performed by several people in collaboration. Such tasks can benefit from the enhanced social interaction and mutual awareness of a CVE. There are certain tasks that seem to be particularly suited to a CVE implementation, and the following three tasks are often mentioned:

1. Prototyping a complex 3D product
2. Training
3. Data visualisation

2.1.1 Prototyping a complex 3D product

VEs should be considered as a useful implementation for product prototyping when:

- The product is complex and 3-dimensional
- The development of a real-world prototype is costly or time-consuming

CVEs should be considered when, in addition to the above requirements:

- The product is developed by several (geographically remote) teams that need to communicate

Caterpillar has recently tested a system developed for collaborative vehicle design. As they have engineers working in different countries, gathering a group of engineers with the required expertise to co-operatively design a vehicle can be quite difficult. In the CVE the experts can meet and talk about 3-dimensional aspects of the vehicle though being physically on different locations. The test was very successful.

From this example we can learn that the use of CVEs in product prototyping can produce the following benefits:

- Decreases time between design phase and market introduction of the new product.
- Allows for improved quality and the design of superior products.
- Engineers can evaluate and test the vehicle prior to building an iron prototype.
- Distributed VR prototyping allows communication of design information to geographically distributed sites.
- Collaborating using video and audio communication within the shared VE helps eliminate poor designs and makes interactive redesign possible.

2.1.2 Training

VEs should be considered as a useful implementation for training systems when:

- The task to be learned is 3-dimensional
- The trainees have to perform several tasks in succession
- Object manipulation is involved

CVEs should be considered when, in addition to the above requirements:

- Several people have to collaborate to perform the tasks

Examples of successful CVE training systems are flight simulators. In these simulators, pilots can be trained to work together on a mission with a specific goal, or they can be trained to fight each other. This is a typical example where VE simulation is needed because a more realistic exercise could easily involve injuries or loss of life, and where the costs of fuel and loss of aircraft would be huge. The collaborative aspect is also obvious.

For VE training systems there are several advantages which make them feasible solutions:

- The trainee can receive direct feedback about the results or consequences of his actions.
- A VE can be structured to reduce the environment-task complexity until the essential relationships are learned. Once elemental task proficiency occurs, complexity can incrementally be added to the training scenario.
- VE provides a considerable degree of control over the self-pacing of interaction.
- VE systems provide sufficient novelty that users can become engrossed.

2.1.3 Data visualisation

VEs should be considered as a useful implementation for data visualisation when:

- Data is complex and multi-dimensional (spatial) in nature

- Data is highly correlated
- Data changes significantly with time
- Data analysis results have to be communicated between people with differing backgrounds

CVEs should be considered when, in addition to the above requirements:

- Actions of individuals influences the combined dataset.

Grand challenge problems in engineering, atmospheric sciences, physical sciences and other fields will tend to produce data results of the type described above. Examples include computational fluid dynamics and molecular analysis.

The Business application, one of the COVEN demonstrators, includes the “business game” with spreadsheets which can be controlled by the participants. The spreadsheets represents production, stock, sale and demand of a product for each month. Productions can be varied by manipulating the spreadsheet. Through this way of data representation and manipulation, the users can see each other’s actions and be aware of the consequences of their actions. Using a virtual environment for a collaborative task gives the users the feeling of being together in the same environment, working on the same task. This increases the quality and efficiency of co-operation.

VEs provide the following benefits in data visualisation tasks:

- Previously unknown three-dimensional relationships can easily be discovered and reviewed
- Measured data can be translated into a three-dimensional reality
- Data analysis can be done with a more natural user-interface
- Data can more easily be manipulated

2.2 Cost and benefits of a CVE for a given task

Here we will give an overview of the factors that need to be taken into account when deciding whether or not to build a Collaborative Virtual Environment for a given task. There are three aspects to this decision:

1. Are the costs and benefits of a CVE implementation properly balanced?
2. How to decide between CVE and other implementations for a given problem?
3. How does the CVE project compare to other possible projects?

To answer the first question, paragraph 2.2.1 will describe the costs associated with developing and using a CVE, and paragraph 2.2.2 will list the possible benefits that may be gained from using a CVE implementation. The second question will not be answered in this document, because too little is as yet known about the benefits of CVE implementations over other implementations for the same problem. In paragraph 2.2.3, we will list criteria that can help to decide between several possible information technology projects within one company, given a limited budget.

Not all factors that are mentioned in the paragraphs below will apply to *every* CVE. The list of benefits for example, is a list of all possible benefits that *could* be derived from the application of VE or CVE technology. For each new CVE, it should be considered which of all these possible benefits can be gained from the application of that particular CVE and in what amount. Moreover, each company can weigh the benefits according to their own standards and preferences.

As a guideline for a new CVE, the costs associated with developing and using the CVE should of course be less than the benefits. A CVE should always offer extra value in a specific situation that balances the costs and disadvantages. So far however, few VE systems have been tested for cost-effectiveness, most VE or CVE systems have been developed as (research) prototypes to which different standards of cost/benefit justification apply.

2.2.1 Costs of Collaborative Virtual Environments

The costs of a particular CVE depend heavily on the complexity of the application, the desired amount of interactivity, the necessary hardware and the desired amount of realism. Demands for each of these factors will force up the price of a CVE. Especially advanced CVE hardware is still very expensive.

Costs for IT projects in general can be subdivided into investment costs and operational costs. Investment costs include:

- shells,
- hardware,
- standard software,
- supporting software,
- runtime software,
- software upgrades,
- I/O devices,
- manpower for development,
- training and testing,
- personnel training to learn new technology,
- office space and materials,
- secretarial support.

Operational costs include:

- maintenance,
- adjustments,
- personnel training to use new technology,
- operating personnel.

Costs can be dispersed when aiming a (C)VE (training) prototype at a set of possible tasks instead of a single application. The use of object-oriented technology and the application of re-usable building blocks may also positively influence the costs of CVE applications.

2.2.2 Benefits of Collaborative Virtual Environments

For most applications more conventional user-interface techniques such as graphic user-interfaces and multimedia will be good solutions. For some applications VR is the best solution because it has the following extra value:

- A feeling of immersion
- Natural interaction and communication
- Addition of a third dimension
- Enhanced social interaction and mutual awareness
- Attractiveness, enjoyability and intuitiveness of 3D images and metaphors

Benefits of a specific CVE implementation can be operational, tactical and strategic. Most benefits may seem difficult to quantify. However, they can be expressed in monetary value, since each benefit serves a purpose, and this purpose can more easily be expressed in terms of money. You have to search for quantifiable components of the benefits. E.g. “job enrichment” will lead to better motivation and less personnel changes, which leads to greater productivity and less training and personnel costs. These costs and benefits can be estimated.

Strategic benefits include the creation of the image of a technologically advanced company when seen to be using forefront technology such as VE, a competitive advantage when first among competing companies to successfully apply new technology, and strategic positioning.

Tactical benefits could include:

- Realise previously impossible things.
- Larger market share.
- In Marketing & Sales Support: better contact with customers, better understanding between customer and salesperson, less time per customer contact, satisfied customers.
- Faster and better quality decisions
- Larger product value and higher quality products.
- Faster time-to-market.
- Work quality.
- Faster revenue growth.
- New product-market combinations.
- Job enrichment & satisfaction, motivation of employees and increased job proficiency.

Operational benefits could include:
- Increased revenues: more sales and higher-value sales.
- Reduced costs: reduced cost of product, reduced costs of sales, reduced management & administrative costs, reduced risk, reduced costs by modelling physical mock-ups that need not be touched in the computer, reduced travelling costs.
- Time savings, multiplied by salary.
- Larger productivity.
- Increased safety.

2.2.3 Comparing projects

The following list of weighted criteria to rank information technology (IT) project proposals stems from the field of information economics. The list contains criteria that should also be considered when deciding whether or not to build a VE or CVE for a given task, especially when the budgetary resources of a company are limited and choices between different IT proposals have to be made.

Possible ranking factors include:
- Return on investment
- Connection with company strategy
- Competitive advantage
- Better management of an organisation
- Consequences of not doing this project
- Fail probability of necessary organisational changes
- How well the system fits into existing IT infrastructure
- Specification of uncertainty in or quality of information needs and system requirements
- Availability of necessary technical knowledge and experience
- Fail probability of new technology

Again, this list is merely an enumeration of possible factors that may influence such a decision, and each company or decision maker may apply their own weights to these factors and decide which factors apply to the current system proposals. For example, for each project, each factor can be given a grade between one and ten, and the total score of a project can then be calculated and compared to the score of other projects. In this case, return on investment can be given a grade moving from ‘1’ when it is negative to ‘10’ when it is more than 500%. However, a company that struggles for survival with competing companies will probably be inclined to attach much more importance to competitive advantage, and will be prepared to take great risks, for example by not considering the last three factors in the list.

2.3 Success and Fail factors of CVEs

Success factors are factors that positively influence the acceptance of a Collaborative Virtual Environment. Fail factors are factors that may negatively influence the acceptance of a CVE.

2.3.1 Success factors

Multi-user real-time interaction is found to be an important factor for the success of a CVE. The additional application of artificial intelligence and the use of intelligent agents makes people feel at ease. The use of metaphors that capitalise on prior skills and knowledge can greatly enhance the usability of a CVE and ease the acceptance. The amount of presence that a user experiences is also a very important success factor.
In collaborative virtual environments, audio communication is believed to be important.
Avatars, also called Virtual Body Representation are crucial in collaborative virtual environments as they represent the point of view of each participant and make one aware of other participants in the Virtual Environment. Section 3.1.1 will focus on this subject.

2.3.2 Fail factors

Fail factors can be divided into four distinct categories: technical limitations, ergonomic problems, socio-psychological factors and risks.

Technical limitations can be divided in limitations of display and interaction. All these technical limitations can put a damper on high expectations of the user. A variety of possible problems are summed up in the next two paragraphs.

Display limitations include: limited field-of-view, poor resolution of the HMD or projection system, time lag and spatial error of the images, distortions introduced by the optics and/or algorithms, frame-rate limitations, perceptual-motor adaptation, and incomplete or conflicting multi-perceptual information.

Interaction problems include: timing lag, spatial error, the kinds and means of interaction and speed. Time lag stems from the fact that users that have different hardware may have a different time frame and this inhibits collaboration (synchronisation), and real-time is more difficult to schedule in a multi-user environment. Speed of the system is a very important factor for users: the speed has to be high enough for users to be able to act naturally and forget that they are in a virtual reality environment (see also chapter 5). Speed can be enhanced in CVEs, e.g. in dealing with network lag: use cheapest protocol for each piece of data, e.g. unreliable protocols for information that is not critical and renewed often, and use a Virtual Place Model where the virtual world is divided into separate spaces and each user only receives information about spaces that are visible.

Ergonomic problems such as simulator-illness, disorientation in real world, social isolation and the discomfort of immersive systems play a role in the acceptance of Collaborative Virtual Environments. Simulator-illness stems from the fact that our body is being fooled by the VE: our vision believes that the body is moving and doing things, while in fact the body may be sitting still in a chair. We want to move our body along with the suggested movements on the screen, reflexes are not correct anymore, depth cannot be seen properly and there are limitations to field of view, sharpness and image frequency. Especially visual signals can lead to Visual Induced Motion Sickness, which manifests itself in nausea at the spot and later, back in the real world, an unstable feeling and disturbances of equilibrium.

The use of VE can also have a significant social impact. The use of Head Mounted Displays isolates a user from the outside world, especially if audio is being used as well. The user will then only see and hear the virtual world, which for some applications is desirable. For other applications however, the application is an enrichment of reality and the use of augmented reality may limit the negative social impact. At present, the comfort of many VE input systems is far from optimal. Head mounted displays for example are pretty heavy due to cables, optical systems, cameras and sensors. The helmet is a heavy load for head and neck muscles as well as for the eyes. Therefore, the helmet can be used for short periods of time only (<15 minutes per session). The helmet gets pretty warm as well and make users sweat. Datagloves are also not very comfortable yet. Many users complain that the gloves do not fit well and makes their hands warm and sweaty.

Also **socio-psychological factors** such as natural resentment, and legal issues play a role.

Risks include:

- newness of technology,
- inexperience in the company,
- use of new tools,
- integration of new technology in existing IT infrastructure
- relatively expensive
- difficult to use
- unreliability of VE systems

2.4 Summary

In this chapter, we have provided information and guidelines for the decision maker, who needs to answer three main questions when deciding whether or not to build a CVE for a particular task:

1. Should I build a CVE for the particular task I have in mind?
2. What are the costs and benefits of building a CVE for this particular task?
3. What factors can influence the project to become a success or a failure?

Guidelines have been provided as to which criteria can be used to determine when a task is suitable for CVE implementation. Three tasks that seem particularly suited to CVE implementation are 3D product prototyping, training and data visualisation. For these tasks, specific criteria have been outlined when to use a CVE to implement this task.

A list of possible cost and benefit factors has been presented. These lists can be used as checklist when determining whether the costs of a CVE balance properly against the expected benefits. In addition to this, a checklist has been provided that can be used when deciding between different information technology projects with a limited budget.
Finally, success and fail factors and risks haven been outlined. Success factors positively influence the acceptance of a CVE by the intended users, fail factors may negatively influence this acceptance, and the risks are inherent to the use of new CVE technology.

3. Guidelines for the CVE designer

The CVE designer translates the intended task into a design for the CVE. In this design he describes how the environment will look like and how the user can interact within this environment. The design also describes how the user can carry out the intended task.

The CVE interface will be a good design when the intended user can carry out a specific task.

During the design process the CVE designer have to be aware of three aspects:
1. What does a user see in a VE?
2. How can a user learn from the things he see in the CVE?
3. How can you make a special CVE?

This chapter will answer these questions. We will not focus on the steps of the design process the designer will have to make in translating the task into the design.

3.1 What does a user see?

3.1.1 Awareness

To be able to identify himself within the virtual environment, the user first has to be aware of his own position, the way he can control the environment with his actions and the feeling of being there, also called presence. For this reason developers have created a virtual body representation (VB) with which the user can identify himself within the virtual environment. With this body the user can interact with his environment.

⇒ Provide the user with a virtual embodiment he can identify with

In collaborative virtual environments (and other kinds of collaborative systems) the virtual body representation becomes even more important. The users ought to be directly visible to themselves and others through a process of direct and sufficiently rich embodiment. (Benford, Bowers, Fahlen, Greenhalgh, Snowdon, 1995). In other words, the users have to feel mutually aware.

By focusing on the functions, behaviours, and relation of the user's physical body to the virtual body Benford e.a. identify a list of embodiment design issues based on their experiences with constructing and analysing a variety of collaborative virtual environments. Some of the more important issues for virtual embodiment design are:

- Location; in shared spaces users need to know their own position and location in relation to other users and objects.
- Identity; users have to recognise who someone is from the embodiment, and be able to differentiate between agents and other users.
- Activity; embodiments should help in conveying a sense of on-going activity.
- Availability and degree of presence; it is useful to convey information about the availability for interaction.
- Gesture and facial expression; gesture and facial expression play an important role in human interaction.
- History of activity; embodiments can support historical awareness of who has been present and what activities they performed.

- Manipulating one's view of other people; to be able to control the view of other people's bodies in order to reduce machine load, and to create subjective views (Snowdon, Greenhalgh, Benford, 1995).
- Representations across multiple media; this has to be considered because embodiment extends itself not only in the graphical, but also in the audio and textual media domain.
- Autonomous and distributed body parts; people can be in several places in one CVE, or in several CVEs at the same time.

3.1.1.1 Mutual Awareness

When designing user embodiments for CVEs we can learn a great deal from observations of people working and collaborating together in conventional settings.

Heath undertook a series of interrelated naturalistic studies concerned with the nature of collaborative work in order to inform the demands for natural, and flexible video-based collaborative work (Heath, Luff, Sellen, 1995). Some of the problems with video mediated collaborative work are similar to problems with virtual collaborative work. The relevant findings for virtual collaborative work can be summarised as follows:

- Co-operative work involves a continuous transition between multiple, interrelated individual and collaborative tasks
- An individual's ability to work on collaborative tasks relies upon peripheral awareness and monitoring of other participants' work
- The participants' activities are mediated and rendered visible through shared objects and artefacts
- Both focused and unfocussed collaboration is largely accomplished through alignment towards the focal area of the shared activity, such as a shared document;
- Collaborative work relies upon individuals subtly and continuously adjusting their access to each others' activities;

For CVE design this means the following guidelines:

⇒ the environment and the embodiments should allow users to switch smoothly between individual and collaborative tasks.

⇒ the embodiment should enable the user to be aware of others and their work, in the environment.

⇒ preferably shared artefacts should be visible and available in the CVE

⇒ the embodiment should make clear where the user has focused his/her attention

⇒ the embodiments should convey a sense of the current activities of its owner

Coven specific collaboration functions are:

- Group navigation should allow a group to follow a leader, human or virtual agent automatically;
- Agents, synthetic humans, should display very human behaviours, essentially guiding the users along;
- Users should be provided with a global map, on which they can locate each other as well as teleport to interesting locations;
- Users should be provided with the option to have subjective views of the environment, selecting to disregard part of the environment, or to highlight part of the environment regardless of the view of the other present participants.

3.1.1.2 Virtual Embodiment Design in Massive - an example

A longitudinal study of MASSIVE-1 revealed that users required some higher level navigation and interaction abilities than currently provided. The wish for improved embodiments expressed by the subjects in the ITW study suggest the need to introduce automatic embodiment behaviours, in order to come to higher level interaction abilities. See Figure 3 for an impression of the virtual

embodiments used in the ITW study. For more detailed information the reader is referred to (Tromp & Snowdon, 1997).

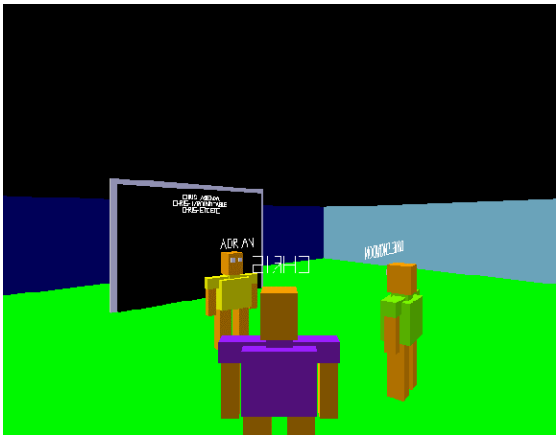


Figure 3: MASSIVE-1 screen shot showing more humanoid virtual embodiments of the ITW study, in front of a “white-board”.

The suggested improvements found in the in the ITW study can be summarised as follows:

⇒ Lock onto an object in order to follow it automatically. For instance, automated tracking of gaze direction to current speaker

⇒ Use behaviour macros. For instance, gestures for waving, nodding, running, walking and facial expressions, and also more higher-order sets of actions, such as the actions a chairperson would want to use to open a meeting

⇒ Objects should react to the user, instead of the user to the object in order to interact with it. For instance, having the object's functions triggered and made available by approaching or selecting the object

⇒ Buttons for often used behaviours should be available. For instance, having an 'applaud' button which produces the sound of hands clapping, the more users press the button at the same time, the louder the applause

⇒ Transparency of control. For instance, it should be obvious who is the current speaker, or who is currently manipulating an object, and a user should be able to go against the automatic behaviour of an object when possible

⇒ Passive capture of often used behaviours. For instance, in a virtual meeting users want to take their usual place around the table, and adjust their view to be able to see all other participants around the table

3.1.2 Attention

The attention of the user is captured by the view and objects in that view straight ahead of the user. Although users will regularly scan their complete field of view, their focus of attention will most of the time be in front of them (Appleyard, Lynch, & Myer, 1964). The quality of light will affect attention too. Light can be used as a resource to direct attention.

There is an optimum viewing distance for examining most objects, depending on the amount of detail desired.

Attention is also concentrated at the points of decision, such as a space with several exits. Exits and other landmarks will take up a large part of the users attention, particularly when they are very distinct (Tromp, 1993). Concentrated attention will also occur when the space from being wide open to constricted or the other way around. These attention directing occurrences can be used as cues in the design of good virtual spaces

3.1.3 Perception of space

Conceptual knowledge of space can be broken down into three broad aspects

- spatial phenomenon
- locational information
- attributional information

For example we illustrate the conceptual knowledge of a person who has described his memories of a `post office' in a Multi User Dungeon, a textual virtual environment.

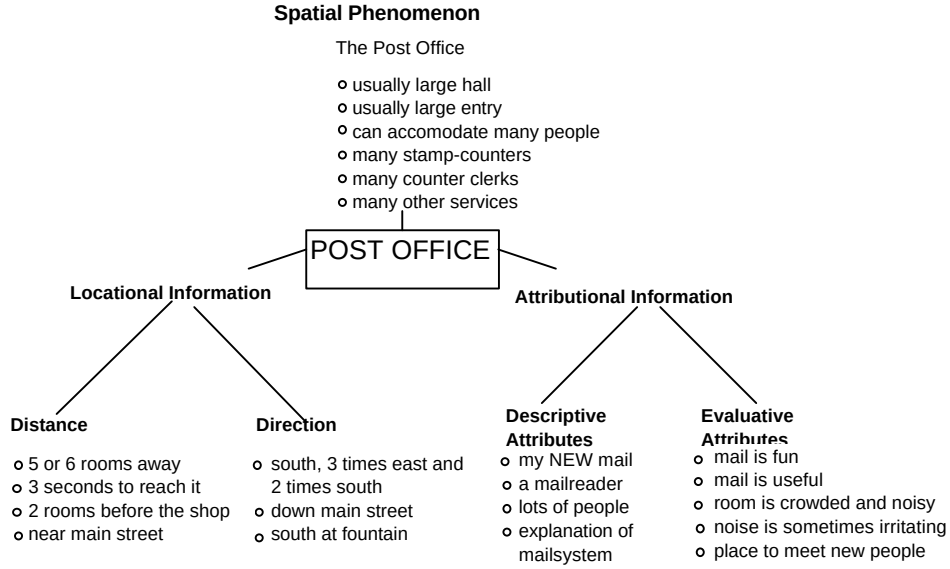


Figure 3 The Conceptual Knowledge Framework, illustrated with the example of a generic post office.

Spatial Phenomenon
The concept post office is a spatial phenomenon: it occupies a place in the CVE, and users understand without much explanation how to use it, and where to find it, because they can associate their conceptual knowledge of a real post office with the virtual one. A concept has all kinds of conceptual information attached to it, such as what a typical representation of that concept is - in this case the typical conceptual knowledge associated with a post office one can go there to mail things, it usually offers a range of services, and it can usually accommodate large numbers of people, etc.

Locational Information
Locational information consists of information about the distance of the spatial phenomenon to other phenomena in the CVE, and information on how to get there from somewhere else. Users typically learn about distance and direction by counting the number of times they have to go in a certain direction to reach their goal. They might learn these numbers by looking around and paying close attention to their surroundings for some time, or by memorising obvious landmarks, the number of turns and rooms to take in certain directions, or by asking an other participant (Tromp, 1993).

Attributional Information
The salience of descriptive attributes contributes to navigation. If a participant has memorised many objects or other obvious landmarks in a room, it should be easier for him to recall where he is, and where he should go from there to reach her destination. Unique features of a room are the source of a rich set of distinctive landmarks, which in turn, increase the number of retrieval cues. Locations often take on significance. For instance, one might remember that in the room with the checked floor pattern, one had to turn left because that is where you met another user who had got lost last time you where there. These evaluative attributes should also increase the number of retrieval cues.

A general guideline can be made:

⇒ Create the illusion of space by replying on all different aspects of knwoledge about space

3.1.4 Perception of motion

When creating the illusion of space and realistic motion in a flat medium effectively, the designer can make use of three criteria:

1. suggest apparent self-motion
2. suggest motion of the visual field
3. use spatial characteristics

1) Apparent self motion
This will be suggested by creating the impression of speed and direction, and by their changes, such as stop-go, accelerate-decelerate, up-down, right-left. Many metaphors can be used to create the illusion of direction; a very large one is the city metaphor (Dieberger & Tromp, 1993).

The visual judgement of motion is based on motion parallax; when moving one's head from side to side the objects one sees are displaced at different rates. Motion is than calculated by the brain from the apparent motion of the exterior objects and is interpreted as being motion in relation to the enclosing spatial form.

2) Apparent motion of the visual field
This will be suggested by creating the impression of passing along-side, overhead, or underneath; rotation, translation; spreading or shrinking of outline or texture; general stability or instability; apparent velocity or lack of it. Even in text-based environments the illusion of motion is easily created, because the user can move freely through the VE (Tromp, 1993).

Users of most of the state of the art CVEs of today will have no real bodily sensations of movement. True kinetic sensations are thus largely absent. Therefore, a user depends on his vision to give him a sense of motion. Apparent motion of the surrounding objects, which the user knows to be really fixed, are interpreted as the result of his own progression. Where surrounding objects are far off, or few, or featureless, or moving with the user, then the sensation is one of floating, of no forward movement.

Conversely, where the near environment has many highly articulated objects, the sensation may be one of great velocity. Details close at hand, such as textures on the wall, rhythm of lights, patterns underneath, all reinforce the sense of speed, according to their frequency and closeness. Acceleration is also heightened on the downgrade, or on sharp turns, while tempo seems to slacken when going upward. The sense of varied motion is inherently enjoyable if continuous, and not too violent. (Appleyard, Lynch, & Myer, 1964).

3) Spatial characteristics
The visual field is interpreted also as a space, a void within which the user can move. The basic sensation of space is confinement to a certain degree. This sensation is modified by in many ways, such as the form, size, and proportions of the space, the kinds of objects in it, and the position of the user in the space. It is also modified by the speed at which one is travelling; things moving by fast become blurred.
Spatial contrast also makes a strong visual impact, and the sequence of spatial changes can be a thrilling experience.

General guidelines to create the perception of motion are:

⇒ suggest apparent self-motion

⇒ suggest motion of the visual field

⇒ use spatial characteristics

The following aspects of motion AND space should be met to create the illusion of space and motion, in order to remain fixed in the mind of the user:

a) The quality of the enclosure; which is based on the presence and position of enclosing objects or surfaces, their solidity and degree of enclosure.
b) The proportions of the enclosure; which are based on scale with respect to the observer, and the position of the observer.
c) The qualities of the light which make the space apparent; which are based on light intensity and direction.
d) The relationship of spaces in sequence; which is influenced by jointing and overlapping of spaces.
e) The direction of principal views which draw the eye toward different aspects of the spatial enclosure.

Thus, the elements which create the illusion of motion and space are the spaces themselves, the objects, motion (of self, and of objects/others), virtual embodiments (self and others). Given unlimited computing- and man-power we could create highly intuitive spaces, objects and functions, which charm people into using and exploring them.

3.2 How does a user learn

3.2.1 Learning from interaction

A mental model of the self and the environment, is built up from the feedback of the interactions of our body with the environment. This is illustrated in Figure 4 motor commands or body movements are co-ordinated by the nervous system. These commands have a results in the world on the effectors or objects in the world. The results of our actions are perceived and fed back to the nervous system. Based on this feedback the nervous system creates a model of the world and the self in this world. When this feedback loop is continuously updated people will experience cognitive immersion. People can become immersed in any new activity and any activity which requires intense concentration, because this creates a continuous feedback loop.

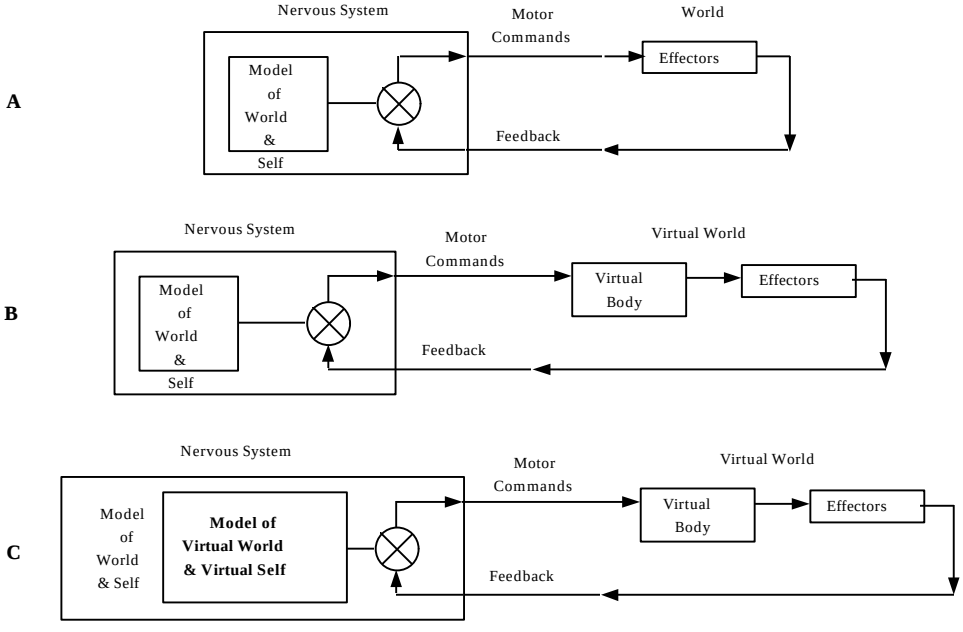


Figure 4 The acquisition of knowledge from interaction with the (virtual) world which leads to cognitive immersion. A) Presence. B) Telepresence. (A and B are adapted from: Loomis, (1992)), and C) Immersion (Tromp, 1995).

We learn by perceiving lawful relations between our actions and their results. For instance, we have learned from experience that dropping a rubber cup will make it bounce, and dropping a crystal ball will make it break on impact.

In order to perceive all relevant information, we should be very aware of our surroundings while learning. We also learn best by receiving a continuous, consistent, broad ranged feedback.

The mental model is refined with time and experience. The more experience a user has the more detailed the mental model has become.

From this theory we can learn for a virtual environment:

⇒ Create a continuous, consistent, broad ranged feedback on the actions of the user

⇒ Create consistency so that the user can build up lawful relations

3.2.2 Learning in a virtual world

Learning from interaction with a virtual world is similar to learning from our interactions with the real world. This means that, if the interaction is implemented in a straightforward, consistent, and believable manner, users will be able to develop a mental model from the VE exactly like they would develop a mental model of a new city they explore. As Tog (Tognazzini, 1992) states in his guiding principles for interface design "The real world is a complex, adaptive structure formed from a few simple, driving principles, and humanity, through millions of years of evolution, has become attuned to its every nuance. If we want our own artificial worlds to communicate most effectively, we will model them after the real world, following its rules wherever appropriate." p. 149.

Perhaps the greatest advantage of virtual reality technology is that it allows us to do things that are not possible in physical reality. This creates a challenge for designers, because it is difficult to imagine the potential of something that has not been invented yet, on the other hand it also holds a potential danger, because we might be complicating the CVE instead of improving it.

Credibility of the happenings in the CVE is not based on realism as compared to the real world, but on realism within the frame of reference of the happening. Things do not have to be real but they have to be realistic to be credible. It is in this way that features and happenings that have no real world equivalent can be made to appear realistic.
To the extent that both worlds are similar we do not need to learn, we can predict the results from interactions in the virtual world on the basis of our knowledge of the real world - our conceptual knowledge. Thus we build up a mental model of the virtual world, our virtual self, and the rules and commands for their behaviours.

To allow the user to build up an mental representation of the virtual world and his virtual self, the interface to the virtual environment and the virtual embodiment should provide the user with simple and consistent commands. The more often the user interacts with the virtual environment, the more experience he will build up.

Interactivity based on the perception of similarity with real world leads to a simple learning curve.

From this, a rule of thumb for enabling the user to create a mental model of a VE system is:

⇒ Create consistent results between user actions in a VE system

In a non-realistic environment an important rule of thumb is:

⇒ Make things credible within the frame of reference of the happening

⇒ Employ realism to aid recognition and understanding of the surroundings and objects

Actions
People perceive the environment in terms of its potential for action. Likewise, participants in a CVE will be able to behave most naturally if the virtual objects and events convey the same affordances as the real-life counterparts of the objects and events. Affordances are functional meanings conveyed from objects to the observer.

A particular design is useful if it has a visible affordance for the goal for which it has been put there. The visible affordance must be generally understandable, and preferably unconnected to cultural meaning. If we create artifacts without real-life counterparts, we make their affordances more intuitive by choosing a metaphor. Consecutively, if the metaphor suggests obscure or incorrect affordances for the task the object is introduced for, the transfer of conceptual knowledge will be severely disturbed. So, when designing an interface we must take into account the needs and abilities of prospective

users, the capabilities and limitations of the technology, and the affordances suggested by the object appearances and metaphors we have chosen (Gaver,1991).

Rules of thumb when designing for action, based on conceptual knowledge acquisition, and the mental model are:

⇒ Make clear what actions are available

⇒ Make clear what the results of actions are before they are executed

⇒ Make clear distinctions between the different actions available

⇒ Give clear indications of the effect the action will have

⇒ Give obvious feedback about the execution of the action once it has been taken

3.3 How can you make a CVE usable?

3.3.1 A usable interface

The Human Computer Interaction (HCI) guidelines can be used fairly straightforwardly for CVE interface design if the design deals with the 2D sections of the CVE interface. The following list of 10 usability heuristics, is created by Jakob Nielsen, based on a factor analysis of 249 usability issues (Nielsen 1994a; Nielsen 1994b):

Visibility of system status

⇒ The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

Match between system and the real world

⇒ The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

User control and freedom

⇒ Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

Consistency and standards

⇒ Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

Error prevention

⇒ Even better than good error messages is a careful design which prevents a problem from occurring in the first place.

Recognition rather than recall

⇒ Make objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Flexibility and efficiency of use

⇒ Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

Aesthetic and minimalist design

⇒ Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

Help users recognise, diagnose, and recover from errors

⇒ Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

Help and documentation

⇒ Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

Even the HCI usability heuristics may not be so straightforwardly transferable to CVEs as would seem at first glance. The HCI usability guidelines have been developed over an extensive number of years, and they have been developed, and tested for and on 2D, or 2,5D interfaces.
At this moment the usability group of COVEN is working on an extra set of usability guidelines for Collaborative Virtual Environments.

3.3.2 Metaphors

Special elements of the CVE-interface, are metaphors. When using metaphors the designer should be very aware of the usability aspects of the CVE.

It may occur that we need to create something in the CVE that simply does not exist in the real world, so that users are most likely not equipped with conceptual knowledge about the phenomenon you are trying to convey. In this case you will have to find an analogy, or metaphor, which matches your phenomenon as closely as possible. In essence a metaphor is a likening of something known to something unknown, to describe it (Lakoff, Johnson, 1980).

Most metaphors used to create 3D illusions are spatial metaphors. The use of a spatial metaphor for a CVE interface seems to have a two-fold rationale. In the first place it allows for the use of physical space as a metaphor for information space. In the second place it creates a life-like environment in which the user can locate and select objects and perform actions. Thus creating a CVE that can be explored in a similar way as we move around in a real space, suggesting appropriate behaviours based on our conceptual knowledge.

Care must be taken that this metaphor does not call forth any extra conceptual knowledge which does not fit the phenomenon you are creating, because this will confuse the user. In case there is no proper metaphor to be found, you will simply end up creating a new concept in the history of CVEs, and there is nothing wrong with that as long as you try to satisfy the associated aspects of conceptual knowledge that people need to build up their knowledge and understanding of the new concept. According to Laurel (1991) "[interface] actions that are quite novel can be effectively represented by establishing causality and probability [..]". Thus, after choosing a metaphor, ask a number of novel users what kind of conceptual knowledge the metaphor calls forth, and what kind of activities it suggest to them. A complex phenomenon will probably require several metaphors to explain it adequately.

One rule of thumb when using metaphors is:

⇒ Find an analogy, or metaphor, which matches your phenomenon as closely as possible

Magic features
When no appropriate metaphors can be found, magic features can be introduced to the CVE interface. Magic features in a CVE are instantations of actions or objects that do not have to behave according to the rules of the laws of physics. They provide us with a means to mediate between existing physical instantations represented in the virtual space and the interpretative instantations of semantic space. Magic features are in fact symbolic features, which can themselves be based on a metaphor, like for instance a "magic door", or a "flying carpet".

Gombrich (1960) proposes that perception is a function of knowledge and experience and that the artistic imitation of reality is the representation of learned interpretations: a sort of visual language. This visual language will come into existence by inventing new magic features to symbolise semantic structure in terms of objects and actions inside the CVE. Adaptation to magic features would thus be a matter of learning. The implication of this is that we should allow the CVE participant to become familiar through exploration with the conditions of the CVE, so that they can form appropriate inferences.

Rules of thumb for designing a magic feature:

⇒ You can use magic features when no appropriate metaphors can be found. Make sure the causality between actions and effects is clear to the user.

3.4 Conclusion

When designing a virtual environment, the designer should take into account the things a user will see, and what the user will think of the things he sees as if the virtual world was a real world. When this will be clear the designer can build a good interface, according to the rules of Nielsen: an interface which is meant for a specific user for a specific task.

3.5 References

Appleyard, D., Lynch, K. & Myer, J.R., (1964). The View from the Road, M.I.T. Press, Cambridge, Massachusets.

Benford, S., Bowers, J., Fahlen, L.E., Greenhalgh, C., Snowdon, D., (1995). User Embodiment in Collaborative Virtual Environments, in Proc. ACM Conference on Human Factors in Computing Systems (CHI'95), ACM Press, Addison Wesley, pp. 242-249.

Benford, S., Snowdon, D., Greenhalgh, C., Ingram, R., Knox, I., and Brown, C. (1995), "VR-VIBE: A Virtual Environment for Co-operative Information Retrieval", *Computer Graphics Forum* 14(3) (proceedings of Eurographics'95), 30th August - 1st September, NCC Blackwell, pp 349-360.

Bowers, J., Pycock, J., O'Brien, J., (1996). Talk and Embodiment in Collaborative Environments, in Proc. of ACM CHI'96, ACM Press.Carr, K.T., England, R.D., (1993). The Role of Realism in Virtual Reality, in Proceedings of 3rd Annual Virtual Reality Conference '93 in London, Anthony Rowe, Wilts, pp. 24-33.

Dieberger, A., Tromp, J.G., (1993). The Information City - A Metaphor for Navigating Hypertexts, recent research paper at the HCI93, Loughborough, Sept 1993, posted to sci.virtual-worlds in October 1993.

Gaver, W.W., (1991). Technology Affordances, in: (eds.) Robertson, S.P., Olson, G.M, & Olson, J.S., Proceedings of CHI'91: Reaching through Technology, ACM, N.Y., pp. 79-84.

Gombrich, E.H., (1960). Art and Illusion: A study in the Psychology of Pictorial Representation, Oxford: Phaidon Press.

Heath, C., Luff, P., Sellen, A., (1995). Reconsidering the Virtual Workplace: Flexible Support for Collaborative Activity, in *Proc. of the European Conference on Computer Supported Cooperative Work* (ECSCW'95), Sweden.

Laurel, B., (1991) Computers as Theatre, Addison-Wesley Pub Comp.

Lakoff, G. Johnson, M., (1980). Metaphors We Live By, The University of Chicago Press.

Loomis, J.M., (1992). Distal attribution and Presence, in: *Presence: Teleoperators and Virtual Environments*, Vol.1(1), MIT, 113-119.

Nielsen, J. (1994a). Enhancing the explanatory power of usability heuristics. Proc. ACM CHI'94 Conf. (Boston, MA, April 24-28), 152-158.

Nielsen, J. (1994b). Heuristic evaluation. In Nielsen, J., and Mack, R.L. (Eds.), Usability Inspection Methods, John Wiley & Sons, New York, NY.

Snowdon, D., Greenhalgh, C., and Benford, S. (1995) What You See Is Not What I See: Subjectivity in Virtual Environments, *Framework for Immersive Virtual Environments (FIVE'95),* QMW college, London, UK

Tromp, J. G. (1993). Results of two surveys about Spatial Perception and Navigation of MUD space., [electronic publication http//www.cms.dmu.ac.uk/ ~cph/VR/JolaPaper/jola.html]

Tromp, J. G., (1995), The Cognitive Factors of Presence, Telepresence, and Immersion: Embodiment and Interaction in Virtual Environments, MSc Thesis, University of Amsterdam, The Netherlands.

Tromp, J.G., Snowdon, D., (1997). Virtual Body Language: Providing appropriate user interfaces in collaborative virtual environments, in: Proceedings of the VRST'97 Conference, Geneva, Switserland.

Tognazzini, B., (1992), Tog on Interface, Addison Wesley.

4. Guidelines for the developer

The task of the developer is to translate the behaviour aspects of the CVE design into an actual implementation. A complication is, that there does not really exist a “theory” which tells the developer how to do this. Moreover, performance limitations of the underlying VE architecture often preclude the developer from implementing a design in a manner which is intuitive, easy-to-use and pleasant to the user. Also, how can we hope to give guidelines for implementation when tasks for CVEs can vary enormously?

In earlier WP2 deliverables [D2.1,D2.2,D2.3], nine generic, i.e., non-application specific, user-level functions for supporting collaborative activities were identified. These nine make up a very basic portion of the functions necessary for a user to execute specific tasks in a CVE.

- mutual awareness,
- communication,
- human actors,
- subjective views,
- resource management,
- object manipulation,
- participants roles and rights,
- group navigation,
- global map.

Since the aim of this chapter is to provide guidelines for how to implement these functions in a CVE system, we’ll present them in a slightly different order. First, we’ll discuss how to implement human actors and subjective views. Good implementations of these two enable many aspects of the other generic functions to be easy derivatives.

Then, we’ll discuss mutual awareness, communication and resource management. These three functions deal with aspects of interaction between users, or between users via an object.

The next group of functions contains the functions participant roles and rights, and object manipulation. Both deal with the interaction between users and objects. The roles of a user determine what rights he has in manipulating an object, or even determine what the object looks like to him. The function “object manipulation” however is more concerned with tools to help with object manipulation for a given task, and in particular, tools for use in a collaborative environment.

The last group of functions extends this notion of specific tools or functions in a collaborative environment. A global map can be an instrument in a single-user VE, but has extra potential in a CVE, e.g., for locating other participants. Group navigation, or more generally, group functionality, discusses implementation of functions that are possible by exploiting an explicit notion of “group” in a CVE. E.g., if you can easily address your own group, then private communication within the group is possible even though group members are dispersed over the virtual environment.

4.1 Human actors

Realistic virtual environments necessitate the natural representation of participants. “Human actors” are objects in a CVE with complex behaviour that represent humans. Realism in representation involves two elements: believable appearance, and realistic movements. In this section we discuss the elements involved in implementing realistic human actors.

Why is there a need for natural representation of participants in a CVE? Typically, the location and direction of the eyes of a participants' avatar represent his viewpoint into the virtual world from which rendering takes place. By making this viewpoint visible to other participants, motion through the virtual world and interaction with other participants becomes easier. Avatar representation is important in a virtual environment, because it combines the following functions:

- **Perception**, a participant may want to see if anyone is around
- **Localisation**, a participant may want to know where the other person is
- **Identification**, the ability to recognise another person
- Visualisation of others' **interest focus**; in communication it is important to see where the person's attention is directed
- Visualisation of others' **actions**; a participant may want to see what the other person is doing and what he means through gestures
- Social **representation of self** through decoration of the avatar; a participant may want to know what the other participants' task or status is.

Using virtual human figures for avatar representation fulfils these functionalities with realism, as it provides the direct relationship between how we control our avatar in the virtual world and how our avatar moves related to this control. Even with limited sensor information, a virtual human frame that reflects the activities of the user, can be constructed in the virtual world; and this increases the sense of presence in this virtual world.

In the rest of this section we discuss

- what is needed to represent a virtual human body,
- how to control a virtual human,
- how can virtual humans be applied in terms of the generic functions for CVEs, and
- what are general guidelines for implementation of virtual humans.

4.1.1 What is needed to represent a virtual human body

To fulfil the requirements of virtual humans (realistic appearance and complex control of gestures), realistic modelling of human mobility is needed. A structured hierarchy, as close as possible to the real human skeleton hierarchy, should be realised to be able to obtain human-like behaviours for the virtual human. The structure illustrated in Figure 5 below is a close image of the real skeleton : each limb of the human body is represented by a link in the structure. Then, the avatar is constructed on this virtual skeleton composed of a set of joints, each joint composed of one, two or three degrees of freedom.

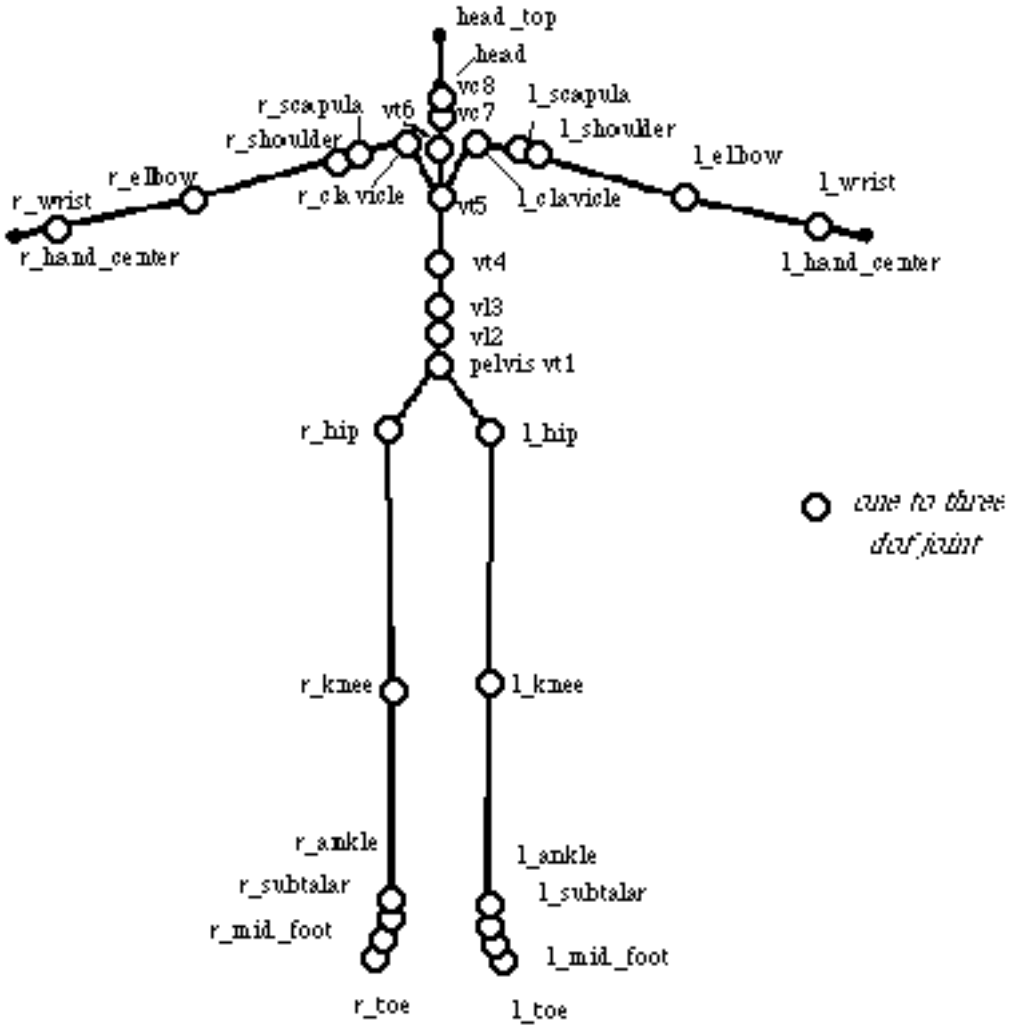


Figure 5 The virtual body skeleton (Joints)

All the skeleton information is based on anatomical data such as segment size, centre of mass, movement limitation (to avoid unrealistic postures of the virtual body). For the graphical representation, a set of surfaces for the body, the face and extremities (hands and feet) are attached to the body hierarchy and each modification on the skeleton position will have its repercussions on the corresponding surfaces.

With such a model, different kinds of avatars can be created by modifying the skeleton characteristics: a tall man or a child, man or woman etcetera.

Current status

This kind of representation is about to be standardised by several working groups which are working in different areas. The MPEG-4 working group is trying mainly to define a way to compress data needed to animate a body based on the previous description. This process will fix the number of joints which are useful for the virtual human representation and the way the information can be formatted to animate this body. The VRML Humanoid working group is trying also to define a generic way to represent a human skeleton in VRML world. They have based their proposal on the MPEG-4 proposal.

Therefore it is likely that a set of rules will be available in the near future which will determine a generic way to build a virtual humanoid by defining the minimum set of joints necessary to represent and animate the virtual body in a realistic way. With such a standardisation effort, any virtual body following this standard will be able to be animated by any animation source following the same standard. Generic libraries can be developed to permit any application to control such body representations.

In the COVEN demonstrators, the representation of such virtual humans has been realized in connection to the two VR languages used in the project, i.e., DIVE and dVS (see e.g. Figure 6). However, these representations have not yet been used in any of the scenarios discussed in the rest of this chapter. This integration will take place in the next phase of the COVEN project.

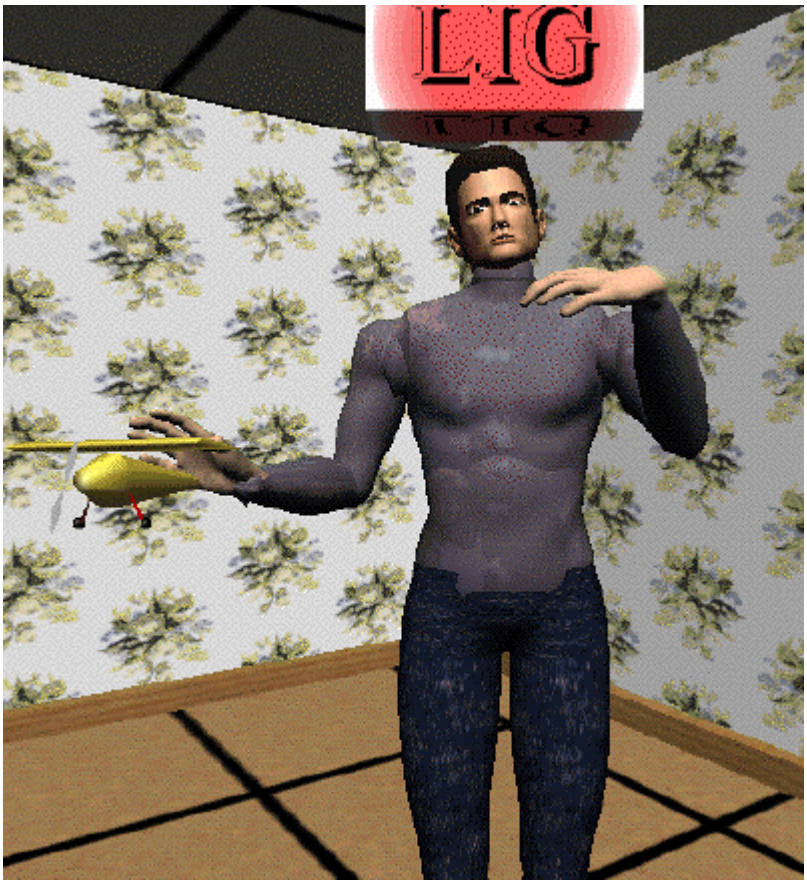


Figure 6 A humanoid controlled by a real user in dVS

4.1.2 How to control a virtual human

The model of a virtual human presented in the previous subsection allows various postures by controlling different sets of parameters :

- **'the global positioning domain parameters'** : these parameters consist of the global position and orientation values of particular observable points on the body (top of head, back of neck, mid-clavicle, shoulders, elbow, wrist, hip, knee, ankle, bottom of mid-toe,) in the body co-ordinate system.
- **'joint-angle domain parameters'** : in the articulated hierarchy, each joint is attached to its parent relatively, with one or more degrees of freedom corresponding to the angles formed in the 3D skeleton.

By changing these 2 sets of parameters, the participant is able to animate his virtual human representation. This animation can be done in real time. But the control of these parameters is not straightforward : the complexity of virtual human representation demands tracking of many degrees of freedom. Interaction with the environment increases this difficulty even more. Therefore, avatar gesture control should use higher level mechanisms to animate the representation with maximal ease and minimal input.

Three different mechanisms are used to control the avatar :

1. The avatar can be an autonomous actor guided by a set of points of interest. The behaviour of the autonomous actor can be made dependent on the set of sensors (such as synthetic view and memory) defining the “ feeling ” of the autonomous actor. This kind of actor can be useful to

make the virtual world much more realistic, for example by adding an autonomous crowd in a virtual public place, as illustrated in Figure 7 below.

2. The avatar can be under direct control of a real person by using magnetic captors, to animate the virtual skeleton. This method of control is a very precise way to animate a body but the devices needed are very expensive and cannot be used by all types of participants. Moreover, this method may also generate an important data flow and slow down the entire application.
3. The avatar can be animated using a set of predefined animations. A menu or list of possible actions lets the user make his virtual representation perform actions such as sitting or walking around, or performing gestures. The implementation can be based on key-frame libraries through an internal driver which is programmed in the avatar to translate these tasks in the appropriate sequence of parameters controlling the skeleton. See Figure 8 for an example set of gestures and postures.

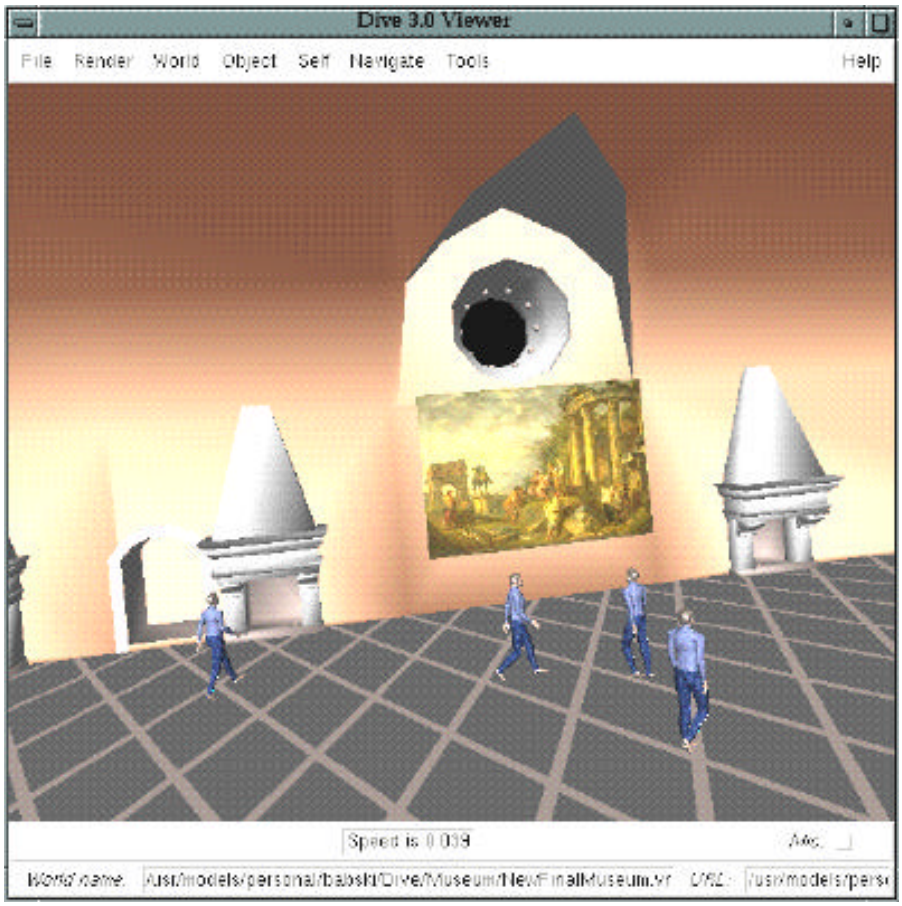


Figure 7 Adding an autonomous crowd in a virtual public place

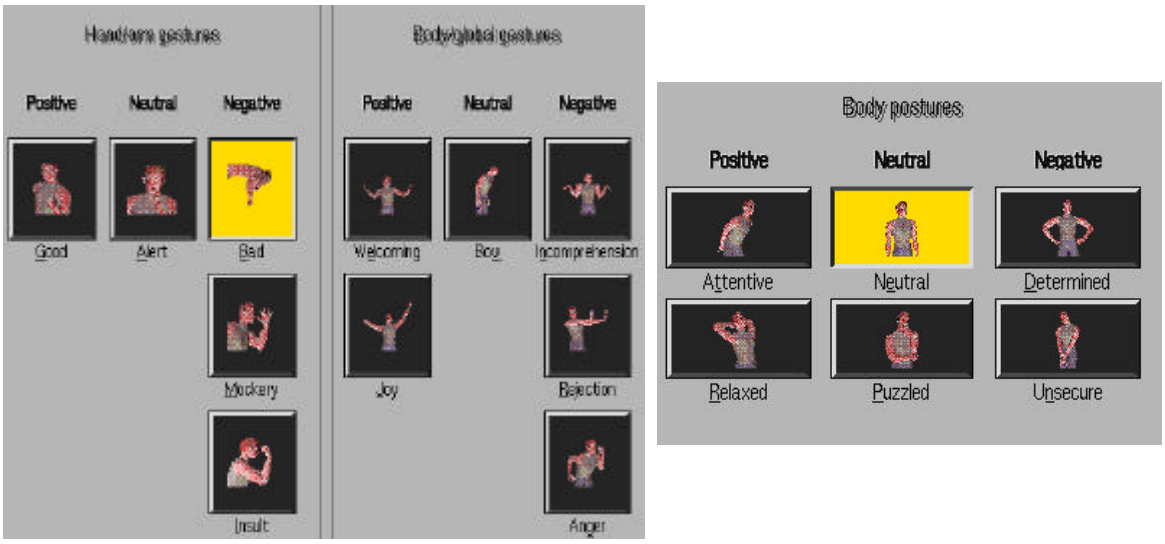


Figure 8 Basic set of gestures and postures

4.1.3 How can virtual humans be applied in terms of the generic functions for CVEs

Whatever the way the avatar is controlled, the action he performs in general supports the following generic functions:

- **Mutual awareness** and **communication**
- **Navigation**
- **Object manipulation**

Mutual awareness and communication

The way a virtual human is animated is very important for the perception and the understanding of other participants of what is going on in the 3D world. Animation should be in direct connection with the action of each user in order to develop the non-verbal communication which is very important in real world. There, gestures and postures have the same importance as speech, and it's very difficult to imagine somebody talking without any movements from hands or head to enhance the meaning of what the person is saying. We should be able to retrieve this non-verbal communication in a virtual world.

Navigation

Navigation means that the virtual human is moving in the 3D world. The simple fact to associate this action by a walking movement, based on a walking motor, can give much more information to other participants than the mere fact that somebody is moving. By noticing the step length, the direction and in a more general way the characteristic of the walk, users can understand the feeling of the virtual humanoid. This is illustrated in Figure 9 below.



Figure 9 (a) Happy Walking (b) Sad Walking

Object manipulation
An avatar can also interact with the 3D environment by grasping objects and manipulating/using them. Such movements may make other participants aware of his actions. This part mainly involves the control of arms (or legs, depending on the application) and extremities, to be able to reach, pick and manipulate objects. This can be realised mainly through the inverse kinematic method.

4.1.4 General guidelines for implementation of virtual humans

The motions of the virtual body and the method to control them will have to be adapted to the specific application at hand. For example, in a sports based application like a football game, the lower part of the body is much more important than the upper part. On the other hand in a business application like virtual banking the nuances of facial expression may be much more important. As mentioned above, each method of control of the virtual body has advantages and disadvantages (cost of equipment, network traffic, availability of equipment).

As a set of general guidelines, the following procedure could be used:
1. Identify from the intended use of the application the consequences for the movements of virtual humans. This should be related to the required detailed functionality of the generic functions awareness, communication, object manipulation and navigation. What kind of movements are necessary? What kind of detailed body is necessary?
2. Analyse the movements in terms of
    - motion complexity (how many degrees of freedom? On what is the motion dependent?)
    - parts of the body involved.
3. This analysis, together with details of the use of the application (costs, hardware available to the user, network traffic, type of user) will determine the type of control appropriate for the virtual human in this particular application.

4.2 Subjective views

In a CVE, the representations of the virtual world may be tailored to particular users within the environment. For example, in an office design scenario the communication cables (normally hidden from view) may be visible to electricians and communication engineers. This both prevents unnecessary detail in the display of some users, while augmenting the displays for the specialists with relevant data. So an important application of subjective views is to allow **task-related views**. But other applications exist as well, e.g., for allowing private communication or changing the awareness of a user.

The requirements for multi-perspective virtual environments involve the manipulation of objects in terms of their presentation to users. From a number of case scenarios, objects within virtual environments should be invisible to a subset of users, while also being visible to others. The nature of

the presentation should also change, for example an object should be able to be emphasised or de-emphasised, possibly by changing its luminance or colour. This is further extended in the case of textual items within virtual worlds, as the contents of the text field may also change. Hence the actual representations of these objects may be different for any number of users.
The implementation of subjective views extends a notion like level-of-detail (see section 5.4.2). While LOD changes the appearance of an object depending on its distance to the viewer, subjective views make the appearance dependent on the *role* of the viewer.

4.2.1 What is needed to represent subjective views

Two independent factors may be extracted from the above description of the needs:
- Differing geometric definition (such as different textual definitions).
- Highlighting and de-emphasising abilities (such as making an object appear brighter)

These two factors, termed 'appearance' and 'modifier', may be arranged orthogonally into a 'view-matrix' which defines an object's range of possible representations.

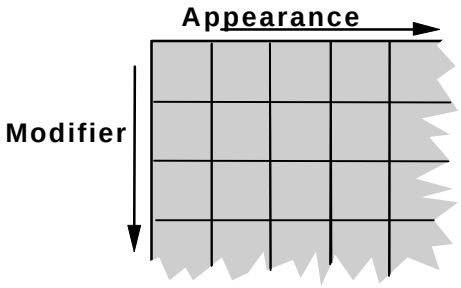


Figure 10 A generic view matrix

The appearance of an object is used to describe its 3D geometry. This may be anything from a simple cube to a more complex building. The modifiers are independent effects that may be applied to any of the appearances. An initial set of modifiers are:
- **Invisible**. The object should not be visible to this user
- **Normal**. Display this object normally, .i.e. no modifications.
- **Transparent**. Render the object transparently.
- **Wire-frame**. Render the object as wire-frame.
- **Dim**. The object should be rendered less luminous (darker).
- **Bright**. The object should be presented to the user and emphasised within their visualisation by means such as increasing its luminosity.

Consider, for example, an object which represents a male human. This may be represented by either a sphere or a generic male human iconic form, see **Error! Reference source not found.**. The modifiers apply orthogonally across each of the representations, allowing one user to view this object as a normal human shaped icon, while another sees a transparent sphere.

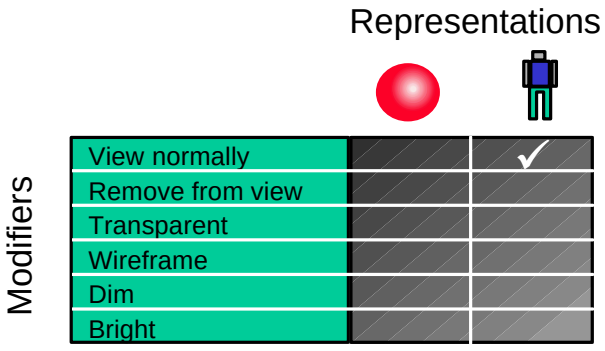


Figure 11 A view-matrix for a male human

This can then be applied, e.g., in a 3D plotting tool, where users are provided with an interface that allows them to choose the representations of the objects in the plot. Figure 12 depicts such a tool

where the user on the left has chosen iconic representations and the user on the right has chosen geometric representations.

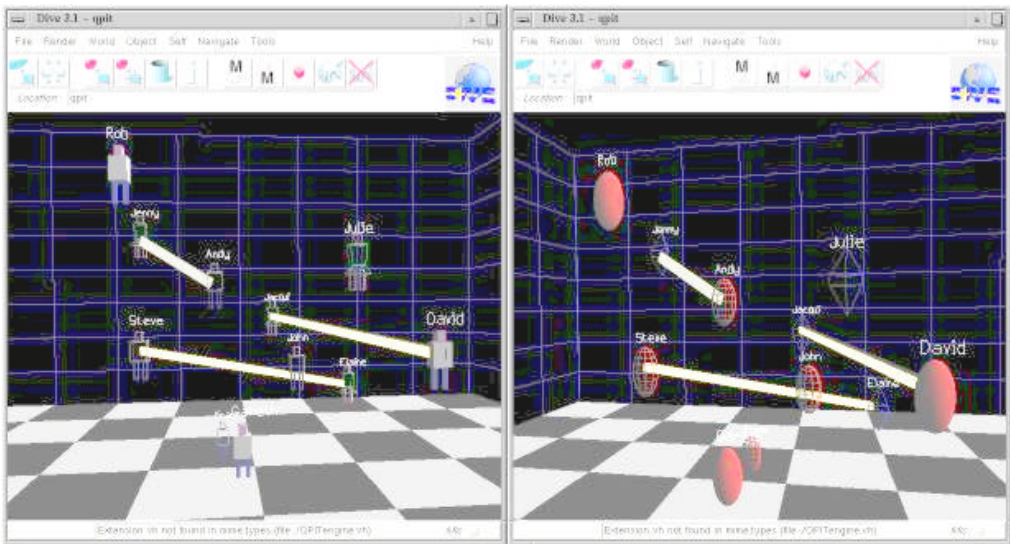


Figure 12 Two user's (subjective) views of the same shared space

4.2.2 How to control subjective views

Controlling subjective views is not very different from controlling other visual aspects of a CVE. We can distinguish between “permanent” and “temporary” subjective views. An example of a permanent subjective views can be found in the implementation of a private board (see 4.4). A private board should be visible to only one user and invisible to all others. This condition will not change during the course of the application, hence it is a permanent subjective view.

An example of a ‘temporary” subjective view is the use of subjective views for indicating a route in the virtual environment (see Figure 16). The route arrows only appear after the appropriate action on the associated object, the global map device, and should disappear again once the route is no longer needed. Here the standard guidelines for a usable interface apply (see 3.3.1), i.e., it should be clear that the route arrows are the result of pushing some button, and it should be clear what to do to hide them.
In subjective views, the actual view presented to the user is dependent on the role of the user. Since a user may be simultaneously in several roles, some sort of conflict resolution mechanism must be employed. Examples of this can be found in 4.6.2, and 4.6.3.

4.2.3 How can subjective views be applied in terms of the generic functions for CVEs

A good implementation of subjective views can support many of the generic functions for CVEs, in particular:

- mutual awareness.
- communication
- navigation (global map)
- participants roles and rights

Examples of the application of subjective view supporting these functions can be found in the rest of this chapter.

4.2.4 General guidelines for implementation of subjective views

The implementation of subjective views requires low level functions in the CVE architecture: like level-of-detail, the image presented to a user is made dependent on certain other status variables. The application of subjective views in other functions has very many possibilities. With any change in

the virtual environment which affects its visual representation the developer should ask the following questions:

- To whom should this change be visible? Usually to the user making the change, of course, to provide a feedback on his actions, but should it also be visible to others? If so, clearly identify the role of these others and why they should see the change.
- Can the change in visual representation interfere with the task other users are executing? If so, the use of subjective views could remedy this, by making the change invisible to them.

4.3 Mutual awareness

Mutual awareness, already discussed in section 3.1.1.1, is an important generic function for a CVE. Here, we shall discuss various implementations in the COVEN demonstrators of its most important aspects. These aspects are:

- Recognition of identity
- Role of participants
- Perception of current activity
- Awareness depending on distance
- Awareness depending on task context

The different scenarios for the COVEN demonstrators put emphasis on different aspects of mutual awareness. In all scenarios it is essential that the user can be aware of the presence of other participants, but each scenario has its own requirements.

4.3.1 Recognition of identity

In the citizen application, where different users may enter and leave, it is important to know who you are currently speaking to, e.g., is this the same person as the one I saw in the previous zone? Also, since one of the goals of the citizen application is to meet other people, it would be desirable if a user could express his identity in the avatar representing him. For example, VR chat applications often allow you to select an avatar from a predefined set.

In the teleconference scenario, a mental model of the ongoing discussion is easier to maintain if it is immediately obvious in their visual appearance who is presenting a new slide, or who is the one voicing a new opinion. For example, the blue guy keeps presenting these boring graphs.

This aspect of mutual awareness can be realised by the proper choice of embodiment for the avatar. Currently, the rather static human body of dVS is used in the COVEN demonstrators, but it will be replaced by a more sophisticated version of EPFL and Geneva at a later stage.

To a degree, both the present human body in dVS and the “advanced version” from EPFL/Miralab are realistic presentations of a human. But this leaves open the question of identity. How to make any avatar personal and recognisable? Some possibilities explored in the COVEN project include

- directing a video camera at the user, and presenting the resulting image on a TV set mounted on top of a blockie body. This would provide a moving image of the user’s head, possibly great recognition plus the added possibilities of non-verbal communication. Disadvantages are, that it only works for non-immersive CVE applications and requires real-time video streaming. The latter is possible in DIVE, but not yet in dVS.
- pasting a static picture of the user’s face on the face of his avatar. This was tried in teleconference scenarios in DIVE, but turned out to be not very recognisable from a distance.
- using personal colours and textures for, e.g., the shirt or the hair on a standard avatar body. Much experience was gathered with this in teleconference scenarios in DIVE. A variation is now used in the dVS versions of the demonstrators.

In order to be able to address communication, it is important to be aware of the other participants’ names. Therefore, name tags could be used. We have chosen to visualise two name tags on the body: one on the front and one on the back, but also name tags above the heads could be used. However, we have already reserved the latter space for future use of “emoticons” (icons expressing an emotion associated with the avatar, such as a question mark indicating confusion, thunder cloud indicating quiet anger, etcetera).

Also, the colours of the body suits have been personalised, enabling recognition at larger distances. This turned out to be a simple, but effective implementation.

4.3.2 Role of participants

In reality, communication is greatly facilitated if it is clear to all participants what their respective roles are. In the COVEN demonstrators there are many examples where a clear awareness of the role of participants can help in executing the task at hand. For example, in the citizen application, if a avatar dressed as a porter is standing near a closed door, the obvious action for a participant will be to address that person to open the door.

The implementation of roles can be easily accomplished in the following ways:

- through the embodiment of participants. Even with the limited possibilities of the present bodies (see 4.3.1), it is obvious which role an avatar has with the name tag “guide”, “travel agent”, “chairman”, “referee”. This only works for roles which are fixed for the duration of the application session.
- through objects (tokens) carried by participants. E.g., in the present version of the citizen application we use a notebook to symbolize a group, and the one who carries the notebook has the role of group leader. It was implemented in such a way, that the simple act of selecting the new notebook attaches it automatically to the user, at a fixed distance but always in his field of vision. This way, he has his hands free for other actions, but is still aware that he has the notebook (and the role of leader). The notebook cannot be detached from him by other participants, only by the current leader himself. Token objects can be used for roles that change.

4.3.3 Perception of current activity

For successful communication, it is important to see what the other participants are doing in a CVE. This “perception of current activity” can be divided into three aspects:

- perception of the focus of attention of other participants,
- perception of current actions by other participants,
- perception of non-activity by participants. It may be that a user is no longer at his desk or even connected, but others can think he is still there, because they see an avatar in the virtual environment.

For the implementation of all these aspects, the embodiment plays a key role. Firstly, the direction of the head of the avatar is an indication where a participant is looking, and can be used as an indication of his focus of attention.

Secondly, in the real world actions by a user are visible by movements of the body. Therefore also in a virtual environment if a user manipulates an object, this should be observed by other users through the movements of the avatar body. However, in a virtual environment it is possible to touch or manipulate an object that would be ‘out of reach’ in the real world. This should also be visible to other users, and there are several possible options for this:

- “stretching” the arms of the avatar so that it still touches the object under control. This may give cartoon-like qualities to the CVE.
- The arm is pointed in the right direction, and a “laser beam” crosses the gap between object and hand.
- A “laser beam” crosses the virtual space between the eyes of the user and the object selected by this user. A similar solution was used rather successfully in DIVE.

At present, the movements of body parts in dVS are still limited and also dependent on the input device. I.e., if you use a space mouse or 3D (tracked) mouse, the movement of the virtual hand reflects the hand of the user in reality. For other input devices it is static. But we note that the movement of the virtual hand should be more a representation of the action the user intended, than what he actually did in reality to accomplish this action.

Thirdly, the perception of non-activity. Although not yet implemented in the COVEN demonstrators, it would not be too difficult to implement a timer on all keystroke or mouse activity by the user. Then, when the user has been inactive for a certain period, his non-activity is visualised both to himself and other users, e.g., by a sleeping cap dropping over his head.

4.3.4 Awareness depending on distance

In the citizen application, when flying over the 3D map of Rhodes, it is not desirable to be aware of other groups doing the same thing, because the task at hand is gathering information about Rhodes. It could be possible to reduce this awareness by making the participants in other groups either smaller, transparent, wireframe, etcetera. When a group is close by, their appearance should be the same as the participants in your own group, in order to facilitate communication.

4.3.5 Awareness depending on task context

The degree of awareness of other participants may be made dependent on the subtask at hand. E.g., in the office design scenario, when you are evaluating the layout of a room, it might actually be desirable to hide all other participants, in case they block your view. On the other hand, in the citizen application where a subtask is to meet possible fellow travellers, it might be desirable to highlight people with the same interests (e.g., determined on the basis of a questionnaire filled out when entering the application).

The examples in 4.3.4 and 4.3.5 have not been implemented yet in the COVEN demonstrators. However, since the awareness in these examples is all subjective to the current user and dependent on more or less fixed roles, we think it could well be realised by using the notion of subjective views (see section 4.2.1 above).

4.4 Communication

In a collaborative virtual environment, communication between participants is essential for executing a task. In the COVEN demonstrators various requirements for communication can be found. For example, in the teleconference scenario, users must be able to talk to eachother, exchange text messages (documents, mail), present images, sound, video. A user must be able to present a 3D-object, e.g. a design, which then can be examined and manipulated by the other users. Non-verbal communication, e.g., through gestures, should be possible.

A general model for communication distinguishes some standard elements:
- the **sender**, the source of an act of communication,
- the **receiver**, the destination of the communication,
- the **message**, i.e., the contents of the act of communication, and
- the **channel** or medium through which the communication is established. A channel is suited for one or more types of communication (audio, video, etcetera).

According to this model, communication is a process in which a sender sends a message to one or more receivers, through a chosen medium. This general model is also used in our current implementation of communication: information is represented by an icon, which is passed from a sender object to a receiver object. For example, if a user wants to share some audio fragment with another user, he must pass an icon (e.g, a CD) to the latter user representing this audio fragment. The receiving user can then use the appropriate device, e.g., a CD-player, to hear the audio. Any type of message can be introduced in this infrastructure, but it will need the appropriate types of sender and receiver.

Currently, communication of text messages and 3D objects is supported. At a later stage, video and audio could be supported. In the present infrastructure, two types of senders were developed:
- the **keyboard sender**, which allows the user to type text messages and send them to other users or receiver objects.
- the **bandybox**, a distributing device for 3D objects.

Three types of receivers were developed:
- the **blueboard**, which can receive both text and objects,
- the **holoview**, which is capable of presenting 3D objects,
- and the **private board**, which visualizes text messages in the user's view.

Using the **keyboard sender**, users are able to send short text messages. While typing, the text appears in the upper part of the user's view for feedback. Text messages can be addressed by typing the name of the addressee, i.e., a user name or the name of a receiver object like a blueboard.

A **private board** displays received text in the lower part of a user's view. Besides the message itself, also the name of the sender is displayed. By associating a user's keyboard sender and his private board with the user name, messages from a certain keyboard sender will be interpreted as coming from the associated user. Similarly, by addressing a certain private board a user is in fact addressing a specific user.

A **blueboard** is a display which can be fixed to a wall or other object in a virtual room. Inspired by the metaphor of a blackboard in reality, a blueboard can display both text and 3D objects. 3D objects displayed on a blueboard can be picked from the blueboard and manipulated by the user.

A **holoview** can be regarded as a blueboard with yet additional functionality. The position of the holoview can be dynamically adjusted by rotating an **axial system** object in the vicinity of the holoview. The holoview serves as a projector for documents, images, video, overhead sheets, geometries, etcetera. It can receive objects from a sender, e.g., a bandybox, and display them in larger size. In case a 3D object is presented by the holoview, a user can look at different sides of the object using the axial system.

The **bandybox** is an object which resembles a mail box, and which is used to distribute 3D objects to receiver objects. Since information can be communicated through handing over token objects, distributing an image or other object to a large group would require handing over copies of the object to each receiver in person. This is taken care of by the bandybox. Sending a text message to the bandybox, respectively colliding an object with it, will ensure that the bandybox sends duplicates of the object to all receiver objects in the zone where the bandybox is located.

All receiver objects have a name, such that they can be explicitly addressed by the senders. In order to address multiple receivers at once, their names can be grouped in a so-called **zone**. The receiver itself is responsible for the action that has to be taken upon receiving an object. For example, when a slide is inserted in the bandy box, a private board may display only the title of the slide, while the holoview displays the graphical content of the slide.

An advantage of text messages on a private board is that it can be used for short (bilateral or small group) discussions without interrupting the main speaker. This has been shown to be very useful. The blueboards have an additional function as a reminder, e.g., in discussions, since they can contain a history of 10 lines. A disadvantage of text communication is that it can only be used in the desk top version of CVE systems, unless the user can type without looking at the keyboard.

In order to facilitate communication between about five users in the teleconference scenario of the business application, we designed the Teleconference Room. The only furniture present in this virtual room is a conference table and four chairs. The function of the table and chairs is to indicate where the users can best take position such that they can see each other and have the holoview in their view. On the table, four blueboards and the axial system are placed. The holoview is placed in front of the table, and the bandybox stands in the corner. The table has a function for orientation and to place objects on. The combination of the table and the chairs makes the Teleconference Room more recognisable. See Figure 13 for a screenshot of the Teleconference Room.

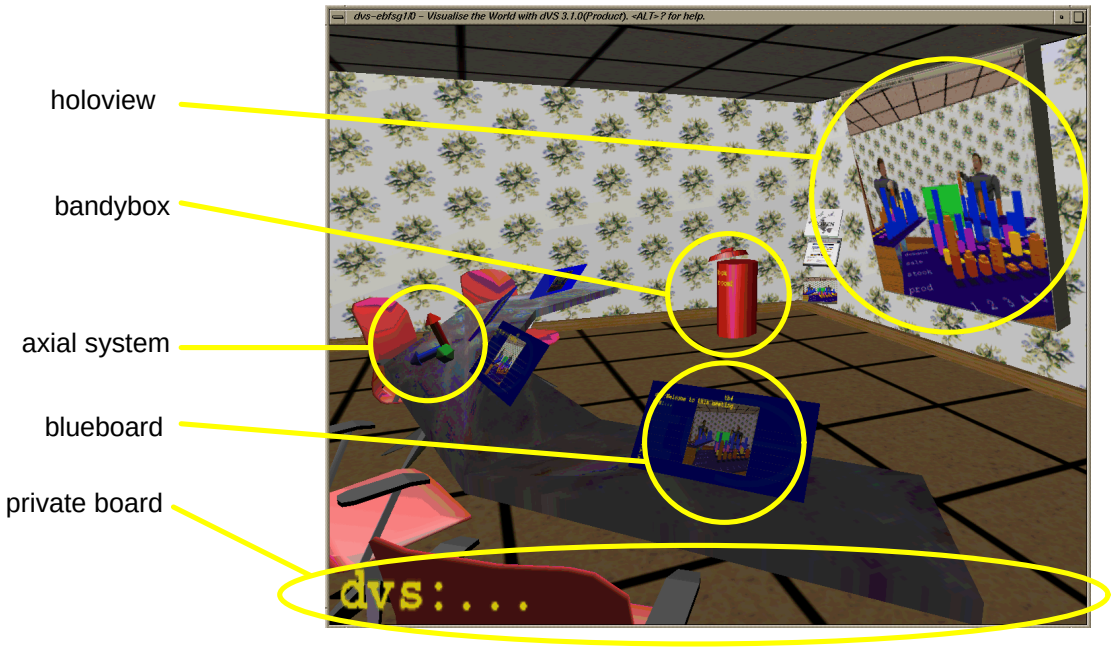


Figure 13 The Teleconference Room

4.4.1 Guideline for developer

Within the general model of senders, receivers and message types, new communication tools can be designed. Inspired by reality, objects can selected for distributing or displaying objects. The scope of a message audience can be limited naturally or explicitly by the user.

4.5 Resource management

Resource management deals with the issues that arise if more than one participant wants to use an object in the virtual environment. No support currently exists within core VR systems to control these *interactions* over objects. Hence external mechanisms, which will eventually be integrated into the core CVE architectures, must be used. There are at least three ways that resource management can be implemented. Which way to use, depends on the circumstances of application and users:

- avoid the conflict over an object.
- let the users solve the conflict as they would in most conflicts in reality, i.e., use social protocols.
- protect the resource.

Below, we shall discuss examples for each method in the context of the COVEN demonstrators, and conclude with some general guidelines for the developer.

4.5.1 Avoiding resource conflicts

Each object in the virtual environment that must be used to execute a given task is a potential source of conflicts in a CVE. For example, in the citizen application we envisioned a flying carpet as a means of transport for a group to fly over the virtual island of Rhodes. However, since the citizen application is targeted at multiple groups of users, and even of groups that have no intention of interacting with other groups, there should be enough flying carpets such that waiting is not necessary. Fortunately, the “laws of nature” in a virtual world allow us to easily create new objects as the need arises. So once a group expresses a desire to use a flying carpet, we create a new one. The expression of this wish is done implicitly, when the group gathers in a teleporter and presses the right button. We must also dispose of the carpet if it is no longer needed: when a group teleports down from the carpet to Lindos, rather than letting an empty carpet fly over Rhodes we delete the object. Recreating a new

one when needed is easier than dealing with possible conflicts over possession of a carpet and "pollution" of the virtual environment.

4.5.2 Use social protocols to solve resource conflicts

Many CSCW systems suggest that arbitrary interaction control, such as locking mechanisms impede collaboration and users' actions within the shared system. Rather, *social* protocols can be used to manage control over shared objects. For example, instead of implementing a round-robin locking mechanism to control access over a device, users should be allowed to communicate (preferably via audio) and manage who uses the device themselves. Requesting use of the device may be as simple as asking "will you pass it to me when you've finished?".

An example of this can again be found in the citizen application. Initially we thought that the control over the flying carpet while flying over Rhodes should be given to a special person, e.g., the leader of the group. This would prevent erratic movements and resolve the conflict of where to steer the carpet. However, since the aim of the citizen application is to explore holiday destinations as a group, we eventually decided that it is up to the group to decide who controls the carpet. Since the dVS/dVISE programming language prevents multiple users from picking the same object simultaneously, it may be possible for one participant to hog the controls. This may be the source of conflicts.

4.5.3 Protect resources through some mechanism

Most interaction conflicts can be removed with sufficient communication between the users. However there are some cases where more explicit resource management is needed. These are when social protocols are not possible, or break down. For example, within a shared space users may place personal items in designated locations, such as a computer which will connect them directly to their file-space, or an address book. These items may contain personal or sensitive information, and are open to abuse by other users, especially when the owner user is not present within the environment. In the citizen application where multiple groups of users are present, you would not want to lose your collected notes on hotels or sites to a malevolent passerby.

Here is it desirable to 'lock' items so that they may only be used by authorised users. Locking an item requires identification of the user, identification of the object and an access model. Support for such a mechanism may be derived from the subjective view mechanism described in 4.2.

An example of a locking mechanism in the citizen application is implemented in the notebook. The notebook is an explicit token for a group and the participant who owns the notebook is considered the leader (at least for the application). The notebook can not be taken away from the leader. Since there should be a mechanism for changing the leader, we decided that the notebook can be dropped (released) by the leader, and picked up by someone else to become the new leader automatically. However, once the notebook is dropped, a casual passerby would be able to snatch it up and become leader of a group he is not even a member of. Therefore, we only allow the notebook to be picked up by members of the group as recorded in the notebook.

4.5.4 Guidelines for developer

In conclusion,

- Examine each object carefully for its potential interaction conflicts given the present task
- See if conflicts can be avoided, .e.g. by duplication of resources in contention.
- Preferably rely on social protocols for the resolution of conflicts.
- In cases where social protocols are not possible or may break down, a locking mechanism could be used.

4.6 Participants roles and rights

In real life, our roles (perceived or authorised) determine what behaviour is socially acceptable. For example, it is not acceptable to take any car on the street and use it, unless you are the owner. In a CVE application, we can use this notion of **participants roles and rights** to guide the interaction between users and objects, thus aiding the user in performing the task he is supposed to do. In a

CVE, the roles of a user determine what rights he has in *manipulating* an object, or even determine what the object *looks like* to him. The latter part is already discussed in 4.2.
Some basic requirements for the implementation of roles and rights in a CVE are:

- It should be possible to give rights to individuals, to groups of people or to everybody at once.
- If a user can have more than one role at a time, there should be some mechanism for conflict resolution
- The role of a user could be displayed in his appearance, thus making it easier for others to interact with him. For instance, it is obvious to others that a user dressed as a porter is allowed to give people access to a building.

4.6.1 Requirements for COVEN demonstrators

In the COVEN demonstrators there are many instances where the use of roles and rights helps the application. The use of subjective views in design applications, such as the office design scenario, has been discussed elsewhere. But also the use of manipulation rights depending on roles can benefit an application. Some examples:

In the business game, where the object could be to maximise group profit by manipulating the individual spreadsheets of participants, one could allow users to manipulate only the spreadsheet of which they are *owner*.

In the citizen application, where a CD player with CDs is used to distribute information about Rhodes, we must not allow participants to take CDs outside the room where the CD player is, or else all CDs are soon lost. Within the room participants should be free to manipulate the CDs, e.g., hand them over to others, etcetera.

In the teleconference scenario, the manipulation of overhead projector and other means of information distribution could be restricted to users in the role of *speaker* or *chairperson*.

4.6.2 Possible implementations

In the implementation for the subjective view mechanism described in 4.2.1, both the **appearance** and **manipulation** rights of objects can be controlled. It is possible to assign users a: **user name**; and a, **user group**. Both are strings which associate users to the properties within objects. A user name is simply the name of the user, such as ‘john’. A user group is an name for an arbitrary role or group of users, such as ‘plumbers’. Groups are used to simply specification of subjective view data, as a single definition can affect many users’ views. An object may have any number of definitions for users and groups, for example:

```
…..
SV_john wireframe                 (user name)
SV_plumbers bright                (user group)
SV_electricians invisible         (user group)
…..
```

Which states that, for example, any user who is a member of the plumbers group will be presented with a transparent view of this object. To prevent the use of complex conflict resolution, a user may only be a member of one group at any one time. Without this constraint, policy decisions would have to be made when two definitions are found. Such as when a user is a member of the plumbers and electricians group (see above), here a conflict resolution mechanism must decide whether to make the object bright or invisible.

Given that a user is a member of only one group at any one time there still exists a conflict resolution problem as a user may have a ‘named’ definition in the same object. For example, in the sample list of properties above, if the user ‘john’ is also a member of the ‘plumbers’ group then two modifiers must be resolved. The resolution mechanism used is that *user based definitions override those of group based definitions*. This permits a group view to be defined, but still allow individual users to configure their display. Hence, continuing the example above, the resultant modifier for John is ‘wireframe’.

A similar set of properties may be used to control access to objects. Rather than use a SV_ prefix (for subjective view), one such as AX_ may be used to denote permissions to access or manipulate

an object. For example, to deny the user John (who will have the access property name AX_john) access to interact with a computer, that object should contain the property AX_john NONE. Other values than NONE may be used to specify more detailed access permissions. Further investigation is required to ascertain the level and type of access control required.

4.6.3 Current design choice

In the present version of the citizen application, some use is made of participants roles and rights. Most prominently, this is done in relation to the roles of a user in a group. For the role of group leader, we have chosen to use an explicit token, the notebook. The user who has the notebook is the leader of a group. This group consists of all the people with him in the teleporter when he pushes the teleporter button. The notebook registers who is the group leader and who are the group members. By using an explicit token, which is automatically bound to the leader at a fixed distance when he first selects the object, it is visible to others who has the role of leader.

A group has two roles: group leader and group member. The latter role is assumed for all people in the group when the group is formed (the group leader is also a group member). This role has certain rights:

- only group members can pick up the notebook if the leader decides to drop it,
- only group members can address in textual communication the entire group without having to address each member individually, and
- only group members can use the personal “return-to-leader” button which automatically teleports them back to the present location of the notebook.
- only group members can become leader of the group (should the present leader resign).
- only group members can decide to leave the group, by pressing the “leave-the-group” personal button.

The role of leader has the added privileges that

- the leader carries the notebook, and therefore represents the position of the group in the virtual environment. I.e., to this position all group members return if they activate their “return-to-leader” button.
- the leader is the only one who can drop the notebook to give up leadership.

Role conflicts are avoided by making sure that a user can only be a member or leader of a single group. Once a user joins another group, he is automatically no longer a member of the previous group.

4.6.4 Guideline for developer

From the description of the tasks to be carried out in the virtual environment, all relevant roles can be distinguished. Also, from the description of the virtual environment, all relevant objects are known. The designer should consider all possible uses of a given object in terms of user input possibilities (i.e., selecting with mouse, picking, carrying the object, colliding with other objects) and consider their consequences for the tasks at hand. This should determine the relation between rights for a given object and the various roles.

4.7 Object manipulation

The function “object manipulation” deals with tools to help with object manipulation for a given task, and in particular, tools for use in a collaborative environment. The participants roles determine what rights a user has in manipulating a given object, but here we deal with how to efficiently support this manipulation.
Types of manipulation include:

- Geometric operations, such as translation (possibly extended with ‘grid-snapping’), rotation, resizing.
- Edit operations are create (possibly subject to a maximum), copy, paste, changing the color, etc.
- Facilities to hand over objects to another user or a group of users. This can be considered a form of communication.

4.7.1 Requirements for COVEN demonstrators

In the teleconference scenario of the Business application, the exchange of information and objects is vital. The use of devices such as the “bandy box” to distribute objects was already discussed in 4.4.

Within the interior arrangement scenario of the Business application a number of users arrange a set of office furniture within a common shared office space. Here, the manipulation of virtual objects is a major part of the task and therefore should be well supported.

4.7.2 Implementation of an object manipulation tool

The following tool is designed to allow the office colleagues to arrange their furniture within the virtual environment, and prevent them from having to haul heavy furniture when testing various possible layouts. In order for users to immersively arrange the furniture the tool allows them to ‘drive’ the furniture around the office. Users can ‘pick’ an object (such as a table) and the *drive* interface appears. See Figure 14 (a more analogue interface is under investigation, which requires 6 degrees of freedom or more for flexible use).

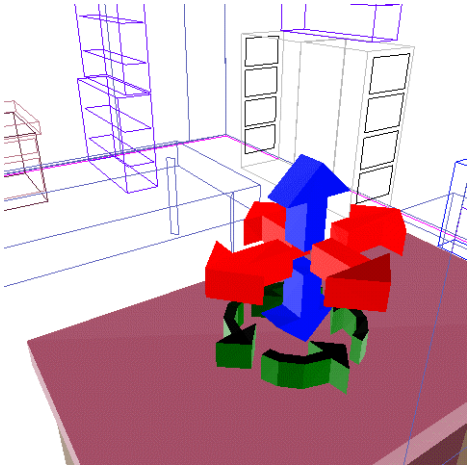


Figure 14 The 'Drive' interface

The objects within the room (such as desks and bookshelves) are provided with simple behaviours, which aid the users when arranging them. Such behaviour includes the ability for objects to notify the user when they are in collision (by turning transparent); automatically ‘snap’ to a wall if it is sufficiently close; and, to ‘fall’ when released onto the object below it.

The action of picking an object can allow other ‘control panels’ to appear, which allow the object to be manipulated, such as the ability to destroy or duplicate the objects; and to view/change the ownership details of the object

4.8 Global map

A global map is an example of a specific tool designed to enhance generic functions in a CVE, in particular awareness, communication, and navigation. A global map can be used for different purposes:

- Showing what is happening in other rooms or places.
- Showing who is where, and thus being used as a visual searching tool.
- As a personal navigation tool, to allow teleporting or easy movement from one location to another.
- As a shared navigation tool, to allow experienced users to show (and draw) the way from one location to another to other users.

Tools as the global map can be necessary in a virtual environment because there, as in regular environments, users have only a restricted awareness of their surroundings. They will pay attention to the different users and reactive objects located within a small radius and will use distant objects as reference points for navigation. The radius of this awareness sphere will change with the size of the environment and its type, i.e., users located in a room will mostly interact within the room, while users sharing an open space, such as a big room or a landscape will have a bigger awareness volume.
In the current section we shall discuss some requirements for a global map in the COVEN demonstrators, a particular implementation investigated in DIVE, the current implementation in dVS, and some general guidelines for developers.

4.8.1 Requirements for COVEN demonstrators

In the COVEN demonstrators, there is a need for a global map in the Citizen application, and in the Travel Training scenario of the Business application. In both cases the virtual environment is big enough such that simultaneous participants may not see eachother everywhere in the virtual environment, or in fact can become lost. All of the functions mentioned above could be applied in a global map in implementations for these scenarios.

4.8.2 A possible implementation

In order to experiment how far it was possible to serve the purposes enumerated at the beginning of this section, a general, non-application dependent, global map, has been implemented in DIVE. When initiated, the global map makes a three-dimensional snapshot of the environment in which it is started, at a smaller scale. This makes it possible to use the map in any environment, immersively or not, and to use it as a helper tool in any application. The global map has the following functionalities:

- It monitors in real-time the different users present in the environment and represents them by simple icons within the map-environment.
- Any user, one at a time, may take the map with him. In this case, this user will be represented differently on the map.
- The map might be iconified and/or scaled up and down, in order to make it fit in all kinds of environments.
- It can be used for navigation by smoothly transporting or by teleporting a user to a given location on interaction.
- A specific mode allows anyone to draw a path on the map. This path is shown on the map and also within the environment using subjective views. The path within the virtual environment can be used in two different ways:
- All users being members of the subjective group are able to follow the path freely, since it is drawn within the environment.
- All users being members of the subjective group are able, on collision or interaction, to follow the path automatically.

The following figure (Figure 15) shows the map in path drawing mode. The map is at the front, at the bottom of the picture, while its control buttons are floating in the air, in the middle of the picture. The regular virtual environment occupies the rest of the picture. It is possible to see a path drawn on the map from the water tower to the church. This path is also drawn in the virtual environment, and can be seen at the background.



Figure 15 Example of a global map in DIVE

In Figure 16, the map is iconified and only represented in space by a floating button. The user is currently following the path from the water tower towards the church. The path is composed of joint three-dimensional arrows.

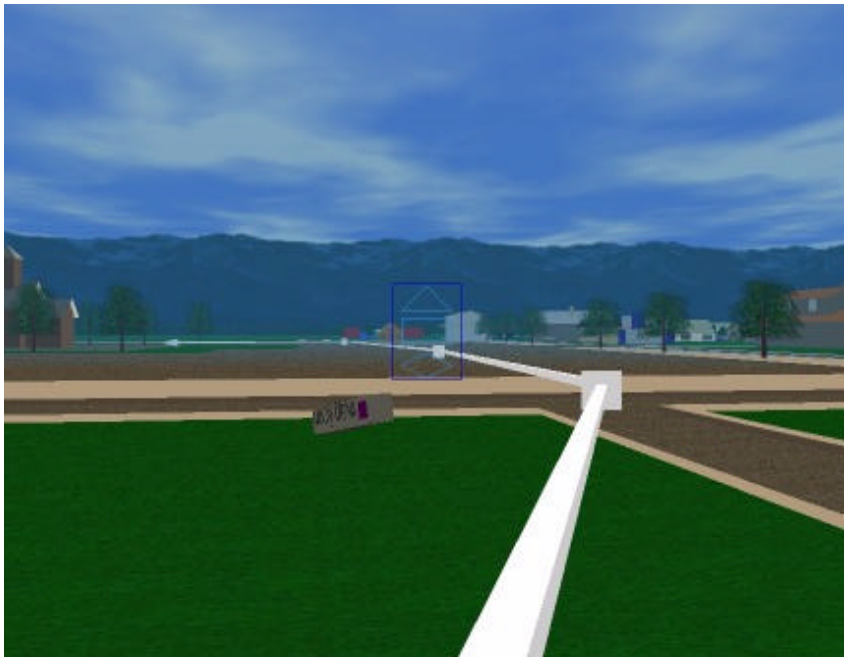


Figure 16 Global map and (subjective) path in virtual environment

4.8.3 Current design choice in the COVEN demonstrators

In the current version of the Citizen application, part of the functionality of the global map is realised through the use of a command language. In the present version, the most reliable form of communication is by using the keyboard, with the resulting text displayed on private board or blueboard (see 4.4). This enables extra functionality, such as required for a global map, since we have a link to the underlying computer system in a way the computer understands. Especially the

second function mentioned above, “who is where”, can be easily realised, by the use of queries by the user. At present, for the global map functionality the following commands are available:

- **?who**, which returns the names of the direct descendants of a zone. These descendants can be zones, names of participants or names of communication devices (blueboards).
- **?Who**, which recursively returns the names of all descendants of a zone. By typing the command a user can find out where all the users are, and also what the names of users and zones are. Then, communication directed to a named zone or participant is possible.
- **?Whereis <name>**, to find the location (zone name) of a given user.

The use of the command language does not yet give a visual representation of the global map. An advantage is, that no object has to be carried by the user, and no provision for resource management has to be made. Moreover, there is no reason why the result of a textual query could not be visualized (just a matter of programming). The main disadvantages of the present implementation are, that the availability of the functionality may not be apparent to the user, and that the precise command syntax may be forgotten.

4.8.4 Guidelines for the developer

From the examples above, we see that designing global maps is application and environment dependent. Also, the appearance of a global map will depend on the different purposes it should serve. Moreover, the CVE architecture may influence the implementation of a global map. For example, if a global map is to be used as an inter-environment spy or navigation tool, it will need specific means to allow it to gather information from the adjacent environments, such as their appearance and the different users sharing them. This is because virtual environment platforms will use the fact that users have a restricted long-distance awareness to cut the "universe" into smaller environments, which the current hardware and software are more likely able to handle (see 5.3).

4.9 Group functionality

Group navigation, i.e., being able to move easily as a group in a virtual environment, is an example of functionality that is possible by exploiting the notion of “group” in a CVE. If the object of a CVE is to execute a task collaboratively, such functions could be a success factor. In terms of the other generic CVE functions, application of group functionality can be found in the areas

- mutual awareness,
- communication,
- navigation,
- object manipulation.

4.9.1 Requirements for COVEN demonstrators

For each of the areas listed above, we can give an example from the COVEN demonstrators:

- mutual awareness: it would be nice if participants from the same group could see eachother more easily in a large virtual environment such as the Lindos zone in the citizen application.
- communication: in the citizen application, since the object is to examine holiday destinations as a group, it is desirable that you can communicate with your group, even though the people in the group are geographically dispersed over the virtual environment. Since there can be multiple groups in the environment, you don’t want to distribute to all groups nor be hindered by their conversations.
- navigation: in the citizen application, you want to be able to fly over the virtual island of Rhodes as a group. Also, you want to be able to come together as a group without having to search for all the others.
- object manipulation: in the teleconference scenario, slides, leaflets or other objects of information should be distributed to the group of conference attendees. It should be possible for attendees to manipulate a 3D object and share the results immediately.

4.9.2 Possible implementations

For the implementation of group functionality, there are basically two questions to answer:
1. What is the functionality you want?
2. How do you tell the computer system what the group is at which this functionality is focused?

The answer to the first question is application dependent and only limited by the imagination. The second question is much harder. In a real environment, what a “group” is in a given situation is usually very implicit, and the result of social protocols. In a virtual environment, the computer system must know exactly which participants are in the intended group, or else it cannot apply the functionality to the right users.

Sometimes the design of the virtual environment can use accepted social protocols to make assumptions on what the intended group is. For example, if a user talks in a room in a virtual environment, the computer system could assume that the sound is only intended for the people in the room. Closing a door can be used as a sign that the group is limited to the people presently in the room.

If a group is geographically dispersed or dynamic in composition, an explicit token can be used to act both as a signal that a new group exists, and to record information on its members. Non-group members can then address this token (or the user in control of it) when they want to join the group, or communicate with the group as a whole.

4.9.3 Current design choice

In the citizen application, we have implemented two types of group navigation:
- **Flying carpet.** This object can be considered a cube of which only one side is visible (the carpet itself). A group of people, explicitly denoted as a group since their names are being recorded in a notebook belonging to the group leader, is attached to this object. The position of the carpet is controlled by a single person, through some kind of “wireless interface”. When the position of the carpet changes, the position of all people attached to it is adjusted accordingly. Independent movement of people on the carpet is allowed, but only within the bounding box of the carpet.
- **Personal teleport button**. Each person has a set of personal buttons attached to his avatar body. When the person becomes part of a group, the personal teleport button can be used to teleport back to the location of the notebook belonging to the group. This can be easily implemented by checking an attribute “belongs-to-group” of the user, then finding the location of the notebook belonging to the group with this name, and teleporting (changing the location of) the user to a location near the notebook.

Notice that group name and group members are explicitly recorded in a database. Group formation occurs rather natural: by together entering a teleporter with the intention to teleport to the flying carpet, the computer system can assume implicitly that this is a group.

4.9.4 Guidelines for developer

Examine the task at hand for identifiable “groups”, preferably groups that remain stable in composition during the task. Examine which of the generic functions listed above (awareness, communication, navigation, object manipulation), involve the group as a whole, while still leaving out other users in the CVE. For these functions group functionality could be useful.

5. Guidelines for the CVE modeler

A major concern when designing virtual environments is to ensure that users can navigate smoothly and continuously through the worlds, with the scene rendered as soon as the users or some objects moves. Ideally, a frame rate of 25 pictures per second is required, but a frame rate of 10 pictures per second is often acceptable to give visual continuity.

Most of the optimisations described in this chapter will help the graphical application to render more efficiently. They will also help heavy operations such as collision detection between objects or intersection of rays and objects. This section is mainly inspired by our experience with the DIVE and dVS systems, but also from other VE platforms. As such, this section tries to remain as general as possible and will not detail the implementation of each platform. However, most of the current VE platforms share a common implementation philosophy and architecture which are used all along this chapter.

This chapter describes the basic techniques to improve rendering, collision and intersection performance. By rendering performance, is meant the number of pictures per second that can be rendered. By collision and intersection performance, is meant the number of CPU cycles necessary to perform these operations. The basic techniques are as follows:

⇒ Simplifying the scene.

⇒ Dividing the scene into chunks in order to help the application performing optimisations. These techniques include spatial organisation and creation of nested hierarchies of objects.

⇒ Giving hints to the application about the shapes used. These techniques include using geometric primitives, using LOD (Level of Details) and setting geometric hints.

5.1 Simplified architecture of a CVE

This section aims at briefly summarising the different bricks which are involved in the graphical and geometrical part of a CVE. These are:

- A hierarchical database of graphical objects.
- A rendering engine
- A (hardware) graphics pipeline
- A collision detection engine
- An intersection engine.

The hierarchical database is the core of the CVE. Typically, a CVE contains functions to manipulate, add and delete objects in the database. Objects contain a translation, a rotation and a scaling vector which describe how they relate to their direct parent, and some children. One of the main operations used within the CVE is the traversal of the database, this operation is time consuming and additional information is traditionally associated to the objects in order to improve its performance. A famous piece of such information is the “bounding box”, two vertices defining a regular box that totally encloses the geometry of an object. Most of the optimisations described in the following sections are related to the traversal of the database, in some way or another.

As pointed out below, the graphics pipeline only accepts polygonal information as input. However, the database can also contain higher-level primitives. The following kind of graphical objects can be found in a CVE:

- Multi-polygons.
- Triangular and quadratic meshes, which are multi-polygons for which the two last vertices of the previous polygon are used to construct the next polygon.
- Cylinders.
- Cones.
- (Regular) boxes.
- (Regular) Spheres.
- Elevation grids.
- Extrusions.

The rendering engine will, as often as possible, traverse the database and send the corresponding geometrical information to the graphics pipeline. During this traversal, high-level primitives are decomposed into pure polygonal information. Graphics hardware and/or software are limited by the number of polygons that they are able to render per second. Thus, the rendering engine will try to minimise the amount of geometrical information sent to the graphics pipeline. Avoiding to send objects which are not currently in the field of view using the objects' position and their bounding box is one of the methods used. There are many more.

The graphics pipeline takes as input three-dimensional transformations, lights and polygons and projects the resulting visible surfaces on the screen's plane, geometrically defined using the current user position and viewing direction. Most hardware and software currently available are optimised for rendering triangular meshes using the Gouraud shading algorithm. They will decompose polygons into triangular meshes at run-time, if necessary. As it was pointed out in the previous paragraphs, this has implications up to the internal data structures within CVE platforms.

The performance of the graphics pipeline in a given scene is commonly limited by either the pixel-fill rate or the geometric transformation rate. You can determine this by re-sizing the drawing context and nothing whether the frame rate changes. If it does change it is probably a pixel-fill rate. This fact should direct the sort of changes mentioned in the rest of the chapter, which, unless specified, help reducing the geometric transformation rate.

Collisions and intersections are the low-level operations necessary for the following functionalities:

⇒ Determine which object a user is pointing at

⇒ Determine if a user has crossed a boundary

⇒ Determine if two objects collide

⇒ Make some surfaces solid, so that users will not be able to navigate through these surfaces and will glide along

Collisions and intersections also involve a traversal of the database, which can cost many CPU cycles. Thus, reducing the number of geometrical tests is also crucial for these operations. One common method is to start by testing intersection or collision with the bounding boxes associated to each object and to continue with the actual geometry only if the test was positive.

5.2 Simplifying the scene

Scene designers are faced with the trade-off between achieving good performance and achieving good visual results. Indeed, achieving good visual results often implies increasing the visual complexity and thus the number of polygons necessary to describe a scene. However, rendering, collision and intersection performance are directly limited by the number of polygons that the CPU and the graphics pipeline are able to handle. This section will help designers to achieve a satisfying equilibrium, using the following techniques:

⇒ Reducing the number of polygons

⇒ Reducing the number of vertices to be sent to the graphics pipeline

⇒ Replacing polygons by textures

⇒ Using appropriate textures

⇒ Reducing light count

⇒ Using special objects, i.e. billboards

⇒ Using two-dimensional techniques, i.e. anti-aliasing

5.2.1 Reducing polygon count

Even if computing power is constantly increasing, low-budget workstations and personal computers are still missing three-dimensional hardware acceleration. In practice, such kind of workstations are only able to visualise 5000-15000 polygons per second, which reduces the number of polygons to be drawn at each frame to 500-1500, if a frame rate of 10 pictures per second is to be maintained. On the other hand, very high performance machines are able to render more than 1 000 000 polygons per second.

Even if it is envisioned that three-dimensional acceleration will soon be available in low-budget hardware, the number of polygons should always be kept low. It is often possible to create models which are visually appealing yet using a small number of polygons. It is often enough to evoke the "idea" of an object using a rough shape. Authoring packages often provide tools to reduce the number of polygons.

It is of good practice to use Gouraud shading to smooth curved surfaces, instead of increasing the number of polygons. Gouraud shading is a method which interpolates the effect of the different lights onto all the pixels composing a polygon, instead of using the same colour for all of them, leading often to faceted results.

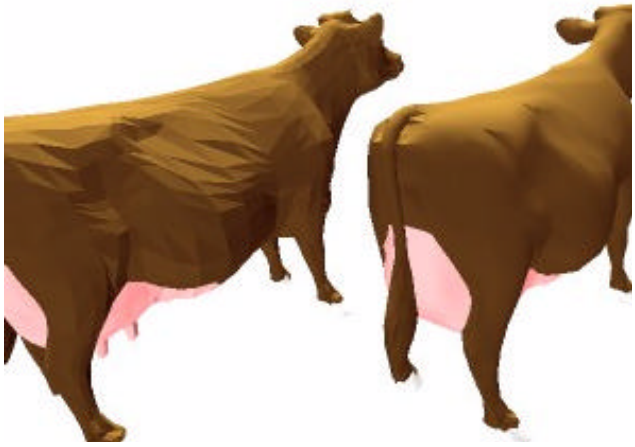


Figure 17 In this figure, the cow to the left is not drawn with Gouraud shading, while the cow to the right is.

Gouraud shading requires to generate a normal for each vertex of a surface. Most VR platforms will create automatically normals by calculating the immediate surface tangent. File formats like VRML allow crease angles to be specified: surfaces will locally be smoothed only if the angle between two polygon normals is under the crease angle associated to a mesh. However, this scheme is not sufficient for complex surfaces, such as those generated from NURBs and it is recommended to use modeling software which supports the edition of normals. It is then possible to minimize the number of polygons used to describe a complex surface correctly or with at a visually acceptable approximation.

Many databases and modelers construct objects out of solids, leaving many interior faces. Even though these interior polygons can never be seen, they will be rendered and re-drawn anyway and will therefore have strong implications on the pixel-fill rate. They will also be tested during collision and intersection checks. Removing the interior faces, either manually, either by setting the correct parameters in the modelers, is one of first optimization that can be performed.

In general, reducing polygon count will not only increase rendering performance, but also collision and intersection performance.

5.2.2 Reducing the number of vertices

Most graphics platforms (including software systems) can achieve much better frame rates when the triangle data passed to them is efficiently meshed. A series of disjoint triangles or polygons requires more vertex processing than a polygon mesh or triangular strip. Certain hardware platforms, particularly high-end Silicon Graphics workstations, like very long triangular meshes (70 is optimal on an Infinite Reality, their best platform!). This is because they have sufficient hardware pipe-lining to deal with large numbers of vertices simultaneously, and disjoint triangles cause the pipe the "hickup" whilst resync'ing to the start of the next polygon. This "hickup" can dramatically reduce performance.

Many models, generated by CAD systems are full of "holes" - these can be caused by numerical inaccuracies during definition and during conversion, and some tools can scan the polygonal surfaces and fix these sorts of problems:

- **cracks caused by close (but not joined) vertices** - may cause pipe hickup, depending on the order of the vertices. Fixing these is called 'welding'. The main effect is probably the visual artifact that might arise from floating point error that causes the polygon to be drawn with a hole even though the error might be minute.
- **very small polygons** - can cause shading anomalies and cost vertex processing time, i.e. reduce the geometric transformation rate. The fix here is to remove the small triangle and weld the gap.
- **triangle facing problems**. A forward facing triangle is defined by its vertices moving in a counter-clockwise direction. If this is not the case, the triangle may appear to be missing when viewed, but is actually there if you turn the object around. A lazy fix is to assign materials to both sides to hide the problem but (as is discussed later) this hurts fill rate. This should be avoided and the vertices defining a polygon should be ordered correctly.
- Another classic pixel fill-rate killer is that of **detail polygons** (like pasting polygons onto the side of a house to simulate windows). Not only does this affect fill rate, but can lead to z-buffer 'beating' (sometimes called 'striping') - the effect is very visual; you see a fuzzy, jaggy blend of the polygon and that behind it. It only adds a few extra vertices to cut a real hole in the background object and fix the detail polygon properly, and the saving in fill rate can often outstrip the extra vertex costs.

5.2.3 Using textures

Although large textures can degrade rendering performances dramatically; a small texture, mapped on a flat polygon is often cheaper to render than drawing many polygons. Here are a few examples:

- Mapping a texture representing tiles on the roof of a house is better than describing all its tiles using small polygons. See the example below in Figure 18.
- Using textures to paint the different external walls of a house is better than using polygons to describe all windows, doors and other details.

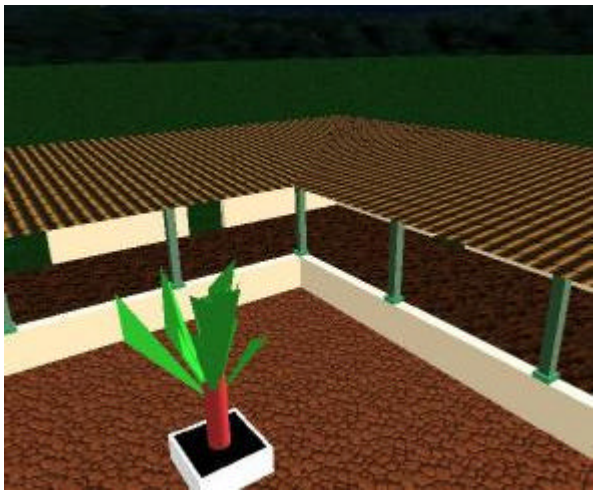


Figure 18 Use of texture

Graphics hardware and software differ greatly in their performances under texture mapping. For some of them, rendering a textured polygon will not take more time than rendering a non-textured polygon. For some of them, textured polygons will decrease considerably rendering performance. Thus, the target platform is to be considered when chopping out geometry and replacing it with texture. Furthermore, there might be a hard limit on texture RAM, making it impossible to render scenes with many textures.

5.2.4 Using appropriate textures

Texture mapping blends the pixels composing the textures with the pixels resulting of a lighted polygon. As such, using one-component textures, which only contain intensity values, is a way of improving rendering performance while producing satisfying visual effects.

In the figure above, the textures used for the floor, the roof and the grass are all one-component textures, blended with an adequate material.

Furthermore, large textures should be avoided, since they degrade rendering performance dramatically on most systems. Textures bigger than 128x128 pixels are too large in most cases.

5.2.5 Reducing the number of lights

The effects of lights on polygons is expensive to render. Indeed, for each vertex of each lighted polygon, the graphics pipeline needs to compute the effect of each light onto the polygon. Reducing the number of lights is then an alternative to increase rendering performance.

Furthermore, the effects of some lights cost more to render than other. Generally, to achieve good performance, it is recommended to favour ambient and directional lights. Spot and point lights should be avoided if possible, as these require much more processing per vertex. To illuminate a scene, two lights are generally enough.

However, lights are often necessary to enhance the visual effects of a scene. Some modeling packages offer some "pre-lighting" solutions based on a method called radiosity. Since most hardware platforms support a limited number of simultaneous lights (typically eight), radiosity is sometimes the only alternative for scenes which require many lights. There are two radiosity methods, exemplified in Figure 19 below. More information can be found in [1], [2] and [3]:

- The first method consists of cutting large polygons into small pieces and calculating, once for all, the effects of the different lights composing the scene at each vertex. A finer lighting granularity is achieved and the effect of the lights is closer to reality on large surfaces.
- The second method applies on textured polygons. The effects of the different lights are calculated for each pixel of the textures mapped on the polygons and the textures end up by being pre-lighted. This method can also apply to non textured polygons, by starting with textures uniformly colored as the underlying polygon is.



Figure 19 Examples of two different methods of radiosity

Pre-lighting solutions create appealing effects and suppress the needs for lights. However, these solutions have some disadvantages:

- The polygon-based radiosity method increases dramatically the number of polygons composing a scene. However, the reduction in dynamic lighting calculations can still result in better rendering performances. Intersection and collision performances will often degrade, since most systems will use the same geometry for collision, intersection and rendering.
- Texture-based radiosity will only offer performance gains on systems on which rendering a texture-mapped polygon does not takes significantly more time than a non texture-mapped polygon. Since this radiosity method does not increase the number of polygons, it will increase all of rendering, collision and intersection performances on such systems.
- Most of the current systems do not allow the lighting effects to be dynamically modified while the application is running.

5.2.6 Using billboards

Billboards are special objects that are rotating around an axis or a point, in order to always face the users. A textured billboard will often offer better rendering performances than a fully detailed object. Billboards, together with partially transparent textures, can be used to draw trees, birds, etc. An example of trees as billboards is shown in Figure 20 below.



Figure 20 Example of trees as billboards

5.2.7 Using anti-aliasing

Anti-aliasing is a method that slightly blends the edges of polygons and lines with the background, in order to make the pixelisation effect, known as aliasing, disappear. Anti-aliasing can help in some cases because it allows to have more chunky polygonal models while cleaning up the edges a bit. However, anti-aliasing needs hardware support to be done properly.

5.3 Dividing the scene

Dividing a large scene into smaller pieces can greatly improve rendering, collision and intersection performances. Division will allow the application to perform some optimizations, by sorting away objects depending on their bounding boxes (i.e., a volume surrounding totally an object and all its descendants in the object hierarchy). These different techniques are the following:

- Creating a hierarchy of objects.
- Arranging the objects spatially.
- Using portals.

5.3.1 Creating hierarchies of objects

Most of the virtual reality platforms work with a hierarchy of objects. The first step to create efficient scenes is to nest objects within other objects. Each object can regroup multiple shapes, which together will make a reasonably sized object.

Each object in the hierarchy will be associated with a (dynamically) computed bounding box, which will be tested previous to any other operation (such as the decision to render an object, the decision to intersect a ray with an object, etc.). In this way, reasonably sized nested objects will help the application to "throw away" uninteresting objects at an early stage.

However, nesting hierarchies too deep should be avoided as well, since three-dimensional matrix concatenation also costs processor cycles.

5.3.2 Arranging objects spatially

When collecting objects into groups and nesting them in a non-flat structure, objects close to each other should be grouped together. For example, all the objects composing one wall of a house should be grouped together, and all the objects composing one of its windows would be arranged in the same group as well. This will help the visualizing application not to draw objects which are outside the field of view.

Building a hierarchy of objects allows to specify the location of an object within its parent. This kind of local positioning should be used, instead of flattening all the coordinates of the vertices relative to the

origin of the world. Again, this will help the application to perform costly operations such as rendering, collision or intersection.

In general, large objects that span large scenes should be avoided. Large objects should be manually cut into sections, in order to facilitate culling. For example, if a room has a large floor, this floor could be cut into smaller sections, one with each of the furniture which is geographically part of that section as sub objects. The walls of this big room should also be divided into smaller objects. However, an equilibrium between cutting large polygons into smaller pieces and the total number of polygons of the scene has to be reached.

5.3.3 Using portals

It is not feasible to represent all the universe into a single scene, instead, you should divide it into smaller worlds and use artifacts to transport seamlessly the users from one world to the other: these artifacts are called “portals”, special objects which, when collided or interacted, transport the user to another world.

This division should use the fact that places which are enclosed into other places can be described in a separate scene as soon as their interior can not be seen from the enclosing scene. For example, when designing a town, you could describe the interior of each house in a separate scene as soon as you can "cheat" an interior for each house as seen from the town. "Cheating" can be achieved by texturing or by inserting a copy with only the most visible details. This kind of setup prevents interaction between users located in the different places. However, since there are obstacles in-between them and since the distance in-between them is long, this is usually satisfactory.

Using portals transfers some of the visibility control onto the application. You can 'hint' to the renderer that these surfaces can't be seen and therefore it need not bother drawing them. The ultimate solution is occlusion culling, where the renderer can detect the surface is behind another before doing the draw - the occlusion calculations may be expensive, but not as expensive as the fill operation. As occlusion culling is seldom used in VR platforms, relying on a good division of the universe into different zones, with portals in-between them is a proven method.

5.4 Helping the VR application

Structuring correctly a scene and simplifying it are two ways of increasing performances. There are other techniques, including:

- Using high-level primitives.
- Using levels of detail (LOD).
- Using geometric hints.

5.4.1 Using high-level primitives

Using high-level primitives such as spheres, boxes, cylinders, ellipses, cones, extrusions, elevation grids, texts, allow the VE application to use more optimized algorithms for rendering, collision and intersection.

As far as rendering is concerned, the applications will be able to decide a way of rendering each of these high-level primitives depending on the distance between the viewer and the object, reducing at the same time the number of polygons needed to achieve a visually acceptable result. For example, a cylinder can be visualized as an extruded hexagon (described using only 12 vertices and 8 polygons) when its size on the screen is sufficiently small. When its size on the screen is big, much more polygons and vertices will be needed to give the impression of a smooth and round object. The following figure shows a detailed cylinder to the left and a simplified cylinder to the right.

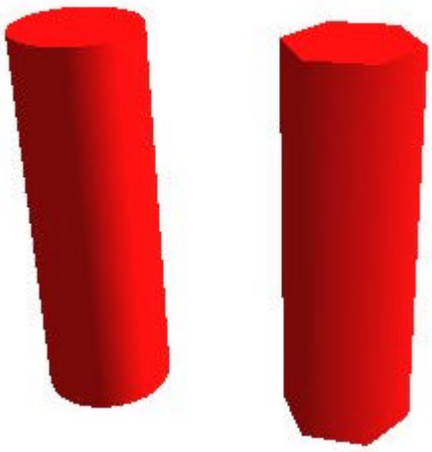


Figure 21 A detailed cylinder to the left and a simplified cylinder to the right

Intersection and collision algorithms will gain precision, since they will be working with perfect geometrical primitives.

5.4.2 Using Level-of-detail

Levels of detail (LOD) are alternative representations of the same object, each with a different level of detail, i.e. usually with less polygons, smaller textures, simpler primitives, etc.

It is often a good idea to use high-level primitives for the less detailed LODs, since most of the applications will provide rendering optimizations for these primitives, as described in the previous section. In some cases, the "worst" LOD can even be an empty one.

Some authoring tools provide automatic LOD generators. The following figure presents an example of such an automatic generation: The pirogue at the top is the original object. Under it, close to the user, is one of its LOD, zoomed to show the reduction of polygon count. Finally, in between both pirogues is the reduced version from a distance, as a user would effectively see it.

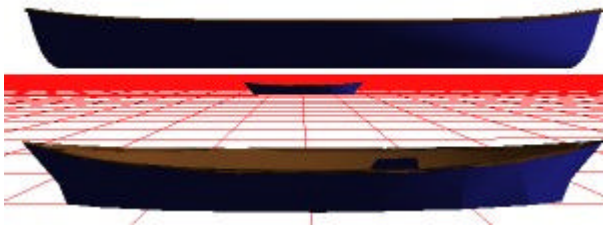


Figure 22 Example of automatic LOD generation

5.4.3 Geometry hints

Usually complex polygons structure can be associated with geometry hints which describes the proprieties of the polygon mesh. Some of these hints can be used by the visualizing application to perform optimizations, some of them will overload the rendering algorithm:

- **Backface Culling:** Most objects in the real world are *solid* objects, i.e., objects which are composed of closed surfaces, and which inside is not possible to be seen. When an object is solid, the visualizing application can throw away all polygons whose normals are pointing away from the viewer. This method is called backface culling and is an efficient way of optimizing performances. Indeed, only the front-facing polygons will need to be rendered at each frame, which might cut the work of the rendering application by a half. The information on solid objects can also be used to turn off two-sided lighting.
- **Convexity**: In general, it is less work for a visualizing application to render polygons which are convex than to render polygons which are concave.

- **Transparency**: Transparent surfaces are surfaces through which it is possible to see to a certain degree. Most of the current systems are optimized to render totally transparent and totally opaque polygons. Objects which are partly transparent will give a hard job to the rendering algorithm and should therefore be use with parsimony.
- **Decals**: Some platforms support 'decals', a method of flagging that geometry is on top (or behind) another surface. Whilst not as general purpose as Binary Space Partitioning (BSP) trees, this can help in drawing certain geometry features (for example, the lines along the center of a road) by hinting to the hardware to disable the z-buffer operation - you have told the hardware it's on top.

5.5 Conclusion

This chapter has presented a series of methods to increase rendering, collision and intersection performances of VR applications. However, there are no hard, fast rules to graphics optimisation. The only workable solution is to try, and try and try again until it goes fast!

5.6 References

[1] Rui Bastos and Michael Goslin and Hansong Zhang - "Efficient Radiosity Rendering using Textures and Bicubic Reconstruction" - 1997 Symposium on Interactive 3D Graphics, ACM SIGGRAPH, 1997.

[2] Michael Goslin - "Illumination as Texture Maps for Faster Rendering" - Technical Report, Dept. of CS, U. of North Carolina, Chapel Hill, 1995.

[3] Reid Gershbein and Peter Schröder and Pat Hanrahan - "Textures and Radiosity: Controlling Emission and Reflection with Texture Maps" - Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24--29, 1994), Computer Graphics Proceedings, Annual Conference Series, pp. 51-58, ACM Press, July 1994.