INTERACTIVE 3-D SPATIAL ANALYSIS IN A VIRTUAL REALITY ENVIRONMENT

S.T. Bhunu 2004

INTERACTIVE 3-D SPATIAL ANALYSIS IN A VIRTUAL REALITY ENVIRONMENT

SOLOMON TICHAONA BHUNU

UNIVERSITY OF CAPE TOWN

Presented to the University of Cape Town
in fulfilment of the requirements for
the Degree of Doctor of Philosophy

Cape Town
2004

DECLARATION

I hereby declare that all of the work contained in this thesis is the result of my own workand is not the outcome of any work done in collaboration. Where data has been provided from other sources, this has been acknowledged.

Furthermore, none of the work presented in this thesis has been or is currently beingsubmitted for any other degree, diploma or other qualification at any other university.

Finally, none of the work contained in this thesis is substantially similar to any otherdissertation, diploma or thesis that I have submitted at another university

Solomon Tichaona Bhunu 20004

"Any solution to a problem changes the problem"

R.W. Johnson

Acknowledgements

The work described in this dissertation would not have been possible without the advice, support and encouragement of many people. First of all, I would like to thank my research supervisors, Prof Heinz Ruther (Geomatics) and Dr James Gain (Computer Science) for helpful advice and discussion. I also would like to thank Samuel Chetty (Computer Science) for the support he gave throughout this research work. To computer science (Graphics research) students, thank you for your sound advice and discussion. Again, to Prof. Heinz Ruther and the rest of staff and colleagues, thanks for comforting me during the loss of my mom, without which, I would not have re-discovered the energy and verve to carry on with my research work. Heinz, you remained a fatherly figure during this period. To my family, you provided me with all the moral support.

I am heavily indebted to Dr Siyka Zlatanova for providing me (through her works "3-D GIS for Urban Development") with invaluable 3-D urban modelling fundamentals and research insights, as well as giving me the data which was instrumental in testing my research theories.

I am grateful to the University of Cape Town for offering me the University Council Scholarship, without which, I would not have gone through with this research work. To you all: thank you

Abstract

Interactive 3-D spatial analysis in a Virtual Reality Environment

The emergence of virtual reality and related tools enables the fundamental infrastructure to begin building virtual cities, which can provide an interactive simulation and analysis environment for planning and management of urban places. The virtual city will provide urban managers with a computer environment to interface with the multitude of complex physical and social data needed to plan and manage cities. A range of innovative technologies are being developed that offer different ways of modelling and representing built-form and associated urban information with real-time interaction over the Internet. For all these efforts in technological development, one of the main topical issues remains, the development of a representation (data structure) that is capable of both *static* and *dynamic* spatial analysis operations.

This research focuses on developing a 3-D data representation for urban management, which would fully support both *static* and *dynamic* spatial analysis operations. It further explores the possibilities inherent in a hybrid of the Boundary representation (B-rep) and distance field modelling (a technique which is finding application in 3-D medical imaging). The research makes an analysis of the existing B-reps before developing the best form which could easily be integrated with the Distance Field (DF). Since this is the first known research in application of DFs in urban GIS, the research further offers the design and adoption of Distance field maps. Further designs are undertaken for the necessary algorithms that would allow dynamic analysis operations to be implemented within the DF environment. The conceptual design is mapped through Entity-Relationship modelling into a Database Management System (DBMS). The B-rep component is maintained within the DBMS whilst the DF component is generated on the "fly".

For the distributed application development, a 3-tier approach that merges the client-side (web browser), application server and database server is proposed. Based on this approach, a Web-based prototype toolkit is designed and implemented using affordable "off-the-shelf" software applications and resources that are relatively easy to set-up and use, and would require standard PC-processor power available to a home user with a modem link (i.e., not a high-end graphics workstation).

The novel aspects of this thesis can be summarised as:

1. The use of a hybrid representation is new in 3-D GISs.

2. The use of Distance Fields and the development of related spatial analysis operations is new in Geo-Information Systems. Furthermore, the research proposes a new distance field modelling approach; the *single Distance Fields* (see Chapter 5).
3. The implementation environment makes use of the existing tools and integrates them in a novel way.

Contents

Figures

Tables

Glossary of Acronyms

3D API:A 3D application-programming interface which controls aspects of the 3D rendering process.

ADFs: Adaptive Distance Fields. Distance fields encoded in a hierarchical structure (see Distance Fields Below)

ASP: Active Server page, a Microsoft programming language for creating interactive web pages using Visual Basic

CAD: Computer Assisted Drafting system: A software used in art, architecture, engineering and manufacturing to assist in precision drawing.

CSG: Constructive Solid Geometry (CSG) is a 3-D solid modelling representation. It represents the solid object as combinations or constructions of solid components (primitives) using the *regularized set operators* and linear transformations: rotation, translation and scaling

DBMS: Database Management System. a software system that facilitates the creation and maintenance and use of an electronic

DFs: Distance Fields: An evolving 3-D modelling technique used mostly in medical imaging

DTM: Digital Terrain Model. A surface representation within a computer environment.

EAI: External Authoring Interface. An API that facilitates user interaction with a 3-D scene generated in a VRML plug-in.

GUI: Graphical User Interface. Interface that allows the user to interact with a model within a graphics package

JDBC: Java Database Connectivity. An API that facilitates the communication of a database and application over some networks

JSP Java Server Page, a programming language for creating interactive web-pages using java programming language

LMC: Line Membership Classification. A spatial query operation which gives the status of a line with respect to an object. The status could be inside, on the boundary or outside.

LOD Level of Detail: A functionality of most 3-D applications and tools that allows the visual of 3-D scene with varying level of detail as a function of the user's closeness to the scene.

mDFs. Multiple Distance Fields. A number of distance maps, each modelling a single object (see Distance Fields, above).

NURBS :Non-uniform Rational B-Spline: A type of spline that can represent more complex shapes than a Bezier spline.

ODBC: Open Database Connectivity. An API that allows an application to communicate with a database (Microsoft's API)

PMC: Point Membership Classification. A spatial query operation which gives the status of a point with respect to an object. The status could be inside, on the boundary or outside.

RDBMS: Relational Database Management system. A database application that uses the relational modelling technique

sDFs: single Distance Fields. Distance Field capable of modelling at least one object (see Distance Fields, above)

SQL: Structured Query Language: A Database language used in relational modelling

VRML : Virtual Reality Modelling Language: A Virtual Reality Modelling language applied to create 3D worlds. A standard for Internet-based 3D modelling, but still evolving.

Chapter 1

Introduction

The emergence of virtual reality and related tools enables the fundamental technology to begin building virtual cities, which can provide an interactive simulation and analysis environment for planning and management of urban places (Ranzinger *et al.* 1995). The virtual city provides urban managers with a computer environment to interface with the complex physical and social data needed to plan and manage cities, along with necessary tools to explore and analyse that data in meaningful and intuitive ways.

The demand for spatial information is most pressing in urban areas (Bric *et al.*, 1994). Most of the population lives, builds, moves about and earns a living in urban areas (e.g.,77.2% in North America, 79.8% in South America, 66.3% in Central America, 74.8% in Europe, 34.6% in Asia, 60.6% in Middle East and North Africa and 38% in Sub-Saharan Africa (United Nations, 2003). The United Nations (2003) projects that the urban population will grow at a rate of four times the rural population between the years: 1990 and 2025. There is thus a need to better manage the urban resources and activities as pressure on the limited structures grows. As a result, interest in 3-D spatial information has escalated (Zlatanova, 2000).

An increasing number of users of two-Dimensional (2-D) Geo-Information Systems (GISs) have been asking for 3-Dimensional (3-D) extensions (Buehler *et al.* 1996, Ranzinger *et al.* 1995, Suter *et al.* 1995, Liggert *et al.*1995).

- Governmental/municipal authorities require improved decision processes. 3-D spatial information and models will mean that they manage the complex road networks, high-rise buildings and telecommunications more efficiently.
- 3-D models also help to improve citizens' participation in decision making processes. With traditional 2-D maps, users are subjected to legends (map keys) and have to recognise the environment through the 2-dimensional "footprints" of features. 3-D spatial information and models will mean that the citizens will be presented with environments they will quickly recognise.
- Scientists and municipal authorities want to simulate the spread of noise, heat and exhaust in big cities.

- Telecommunication companies need 3-D data to establish wave propagation in urban environments.
- Architects want photo-realistic models of existing buildings to plan new ones and to visualize the new designs or additions.
- Vendors of traffic navigation systems need 3-D data to improve the accuracy and ease of use of their systems. These vendors use the Global Positioning System (GPS) and satellite signal obstructions are an issue of major concern.
- Tourism agencies want to offer virtual reality models of destinations that they support.
- 3-D data is also necessary for a number of new applications, which have not been possible using 2-D data (e.g., all kind of environmental simulation systems, disaster prevention, emergency training, etc.).
- An obvious advantage of 3-D over 2-D GIS is increased realism. This not only makes the user interface easier to use and reduces training times for professional GIS users, it also makes GIS data accessible to new groups of users with no previous GIS experience (e.g. in public information systems). Thus, expensive data can be used more widely.

The field of 3-D modelling is quite diverse and an introduction is given below to areas which are of interest to this research work:

1.1 3-D modelling concepts

Figure 1.1. 2.5-D terrain model of the famous Mount St Hellenic volcanic monument. Note the absence of "overhead-shelter" caves.

There are basically two different ways of looking at representations, which take into account the third dimension (normally called the elevation dimension in GIS):

- The most commonly recognized is a data structure where a z value (normally elevation) is recorded as an attribute for each data point (x,y). These z values can be used in a perspective plot to create the appearance of 3-D scenes This is not "true" three-dimensional modelling and is often referred to as "2 1/2 dimensions". These 2 1/2-D plots are an attractive way of displaying topography and other continuous surfaces from Digital Elevation Models (DEMs). As can be seen from Figure 1.1, the surface presented lacks "overheard shelter" caves, which highlights the limitations in 2.5-D for surface modelling. Height is modelled by storing a Z value at each point
- "True" three-dimensional representations store data in structures that reference locations in 3-D space (x,y,z) (see Figure 1.2 Below). Here z is not an attribute but an element of the location of the point. This permits data to be recorded at several points with equal x and y coordinates.

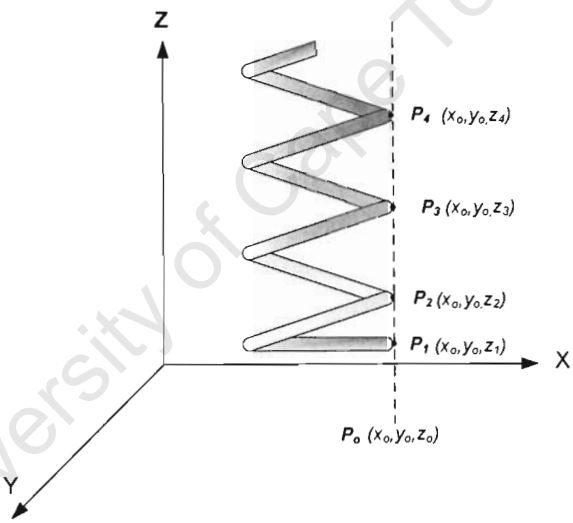


Figure 1.2: Illustrating "true" 3-D modelling. Showing points P_o,P_1, P_2,P_3 and P_4 with the same x, y values but different z elevation values.

1.2 Virtual Reality (VR), Augmented Virtual Realty (AR) and Visualisation

Isdale (1998) states: "Virtual Reality is a way for humans to visualise, manipulate and interact with computers and complex data". In terms of interaction with virtual environments, there is a distinction between immersive or non-immersive VR (MacEachren *et al.* 1999). In this study desktop VR, a kind of non-immersive VR, is employed. Immersive systems are synonymous with headgear, which is worn in order to give the user the feeling of being part of the "reality" and also

to block out normal intrusive reality. These VR systems completely immerse the user's personal viewpoint inside the virtual world.

The term "visualisation" has been used extensively in many fields: from scientific and engineering visualisation to the entertainment industry, from physics to molecular behaviour, to medical visualisation. This research work will apply the definition of visualisation as (Schroeder, 1994) "a process of exploring, transforming and viewing data as images to gain understanding and insight into the data". Whilst VR will provide the means for interacting with the object, visualisation will ensure a realistic picture to the object.

Virtual Reality provides us with the scope for enhanced exploration because it includes motion and interaction within the visualisation rather than just static scenes. Interaction is fundamental to successful 3-D VR systems. The process by which the user explores correlates and comprehends spatial data is by its nature interactive. Users benefit from the ability to interactively modify parameters in their problem domain and to observe the effects in real time. A VR user-interface provides the potential to "zoom-in/out" from a large area to smaller areas of interest which have higher –resolution data, as well as to "fly" over the landscape searching for features of interest (Brown *et al.* 2002).

The virtual reality (VR) concept is increasingly used in varied areas (see Figure 1.3), including urban planning and architectural applications, surgery, archaeology, the environment, statistical data visualisation, GIS and military applications.

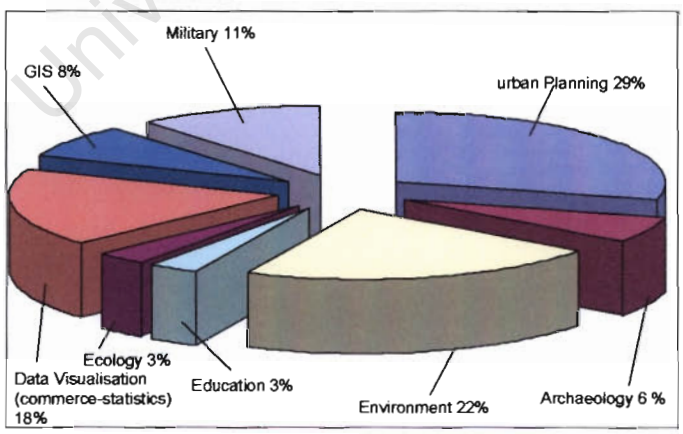


Figure 1.3 Virtual Reality-related projects by research area [adapted from Haklay, 2002]

Analysis of Figure 1.3 above, shows that VR is finding more use in urban planning related projects. Applications in this domain range from attempts to produce large-scale urban models (Liggert *et al.*1995), to explorations of collaborative environments for design and planning (Dodge *et al.*1997). Examples of VRGIS implementations include the prototype for Zlatanova's 3D GIS and Venue project.

Although a lot of work has been implemented in the area of Virtual reality and urban modelling, the same cannot be said of Augmented Reality (AR) (Cain *et. al.* 2004). The basic idea of augmented reality is to superimpose virtual (computer generated) objects over a real-world environment in real-time. The following can be summarised as the most important properties of AR (Hedley *et. al. n.d.*):

- combines real and virtual objects in a real environment;
- Runs interactively, and in real time; and registers (aligns) real and virtual objects with each other. What augmented reality attempts to do is not only superimpose graphics over a real environment in real-time, but also change those graphics to accommodate a user's head and eye movements, so that the graphics always fit the perspective. Here are the three components needed to make an augmented-reality system work:
    - head-mounted display
    - tracking system
    - mobile computing power

In AR, there must be a mechanism to combine the real and virtual that is not present in other virtual reality work. Developing this mechanism is an active research topic.

1.3 Geo-Information Systems (GIS), Virtual Reality (VR) and the internet

The integration of GIS, VR and the Internet is made possible by the use of the Virtual Reality Modelling Language (VRML), an ISO standard for creating interactive 3-D objects and worlds to be experienced on the WWW (Carey and Bell 1997). It is the existence of this standard and the widespread availability of VRML plug-ins, which makes Web-based 3-D visualisation feasible (Brodlie *et al.* 2002). GIS, VR and the Internet can be combined in different ways to support Web-based modelling; visualisation and analysis (see Figure 1.3 below). This is reflected in various VR applications in urban environments and the "3-D" city-scape (Martin and Higgs 1997, Fairbairn and Parsley 1997, Doyle *et al.* 1998). Arising from these applications, a number of GIS-

VR prototypes have also been implemented. The Internet is a network with millions of computers interconnected through various forms of telecommunication, providing infrastructure for information dissemination.

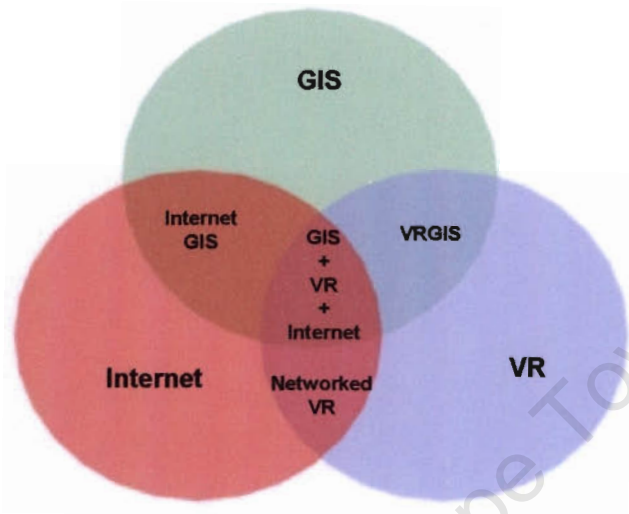


Figure 1.3. GIS, VR, Internet and their integrations

The concepts that arose from combining the three areas: GIS, VR and the internet are briefly explained below:

Internet GIS: The Internet, in particular, the World Wide Web (WWW), has experienced an astronomical growth in recent years, and the Web is now widely used as a distributed computing environment. The GIS community has embraced this, and there is an increasing spread of Internet GIS (or Web GIS) (Plewe 1997). Internet GIS combines the Internet and GIS. That is, a conventional GIS using the Internet as a basic information infrastructure for spatial data dissemination. Because of the nature of the Internet, Internet GIS is regarded as an interactive, distributed, dynamic, cross-platform and client/server computing system, and it has the capability to access various forms of GIS data and functions in an interoperable environment (Plewe 1997, Peng 1999).

In the commercial market, several GIS vendors have dynamic Internet mapping systems within their product range. AutoDesk have Map Guide, ESRI produce MapCafe and Intergraph and

MapInfo have their own proprietary applications. While not designed to produce extensive analytical capabilities these applications are empowering the Web community by making spatial data more readily available. The functionality is at a level where pre-determined data may be queried and examined so that the typical end user is not overwhelmed by unfamiliar (Doyle *et al*, 1998) technology while still capable of learning about, or contributing to GIS discussion over the Web.

VRGIS: VRGIS can be defined as a "traditional" GIS - with its data storage and handling capabilities, and query and analysis capabilities, but with Virtual Reality as the main interface and interaction method (Faust 1995, Raper *et al.* 1991).

It has to be noted, that while VR is inherently 3-D, most existing GIS implementations, are 2-D. As a result, current GIS is mainly 2D and therefore the transition to VRGIS is not transparent. So what is the future direction in the development of VRGIS? Faust (1995) described the ideal VRGIS as having the following features:

1. Very realistic representation of real geographic areas.
2. Free movement of the user within and outside the selected geographic terrain.
3. Standard GIS capabilities (query, select, spatial analysis, etc.) in a 3D database.

In the light of this view, it is clear that such systems do not exist yet, an issue that is likely to be addressed by this research work.

Networked VR: Before the advent of the internet, VR models were standalone, as is the case with most CAD models. Nowadays, with the development of Internet technology, these VR models can be networked, and participants can be involved in VR by logging onto a network computer. A typical example is ActiveWorlds, which has received attention in urban planning (ActiveWorlds, 1999). Users can interact with VRML worlds on the Internet by using the VRML browsers. Some examples in the context of geographical applications can be found in Brown *et al.*, (2002) and Moore *et al.* (1999). More details about networked VR and associated techniques are given by Singhal and Zyda (1999).

A full integration: A full integration provides much advantage for setting up a platform for distributed spatial decision-making. In such integration, GIS provides spatial data and analysis

techniques, which are currently missing in Networked VR, VR helps to visualize the large volume of data in realistic format, and the Internet facilitates information dissemination.

Distributing the application between client and server across the Internet enables remote access to the information in ways not easily possible with conventional applications. Changes in server-based information can be made immediately and be universally available. Simultaneous collaborative sessions are possible. Access control provides a mechanism for creating multiple user classes having different access rights to the information.

The remote visualisation of multidimensional information via the Internet also has many advantages, notably the lack of need for specialist local software beyond a low-cost browser. There is also the potential to search and filter out relevant datasets from databases much larger than could be stored locally (Abel *et al.* 1998).

1.4 Motivation

The research motivation of this thesis is hinged on the challenges of establishing a 3-D data structure (representation) that is capable of 3-D spatial analysis operations and could be implemented in a distributed environment.

A Data representation is a very significant component of any GIS application. For computational processes and analysis operations to be effectively carried out, the world features must be mapped into a data representation.

3-D representations of geometry provide the foundations for various areas of computing that involve geometry. Areas of 3-D modelling include computer graphics, computer aided design, scientific visualisation, 3-D geographic information systems, finite element analysis robotics, computer vision among others, and each of these areas has seen developments of a number of 3-D representations (see Chapter 3) (Naylor, n.d). This can be partly attributed to the fact that different 3-D modelling areas have focussed on varying aspects of the 3-D modelling domain. For example, Computer aided design (CAD) systems are much more concerned with solid modelling, 3-D VR systems are interested in visualisation, hence they employ *extrusion* data representations. *Bezier* surfaces are concerned with surface capture, as are the *boundary* representations. Most GIS products can process topographic data and create 3-D impressions, usually as digital

elevation models (DEMs), and display isometric views or contour maps, in all these cases, the elevation or z co-ordinate, is a single-valued, dependent variable. Hence the representation used is 2.5-D (Weibel and Heller 1991). Another attempt to create 3-D models is made through the use of data layers with explicitly 2-D topology recorded per layer, e.g., ArcCAD, ESRI. In this approach, 3-D spatial analysis between layers creates problems. The area of hybrid representations has not been fully explored as the probable solution to the development of a representation that combines the various strengths of different representations.

Whilst 2-D spatial analyses are well studied, research on 3-D spatial analyses remains an area of major concern (Zlatanova 2000). As with 2-D GISs, 3-D GIS should be capable of performing topological queries (neighbourhood, meet, containment, etc.), metric (distance, length, area, volume, etc,), merging, buffering and feature, classification operations (see Chapter 5). 2-D spatial databases and related interactive querying tools have been developed, and the same is expected for 3-D GIS.

A limited number of tools for 3-D data manipulation can be found in computer graphics and CAD systems. Such systems, however, cannot readily be applied to handle geo-information. CAD systems aim at the design and visualisation of spatial objects. Most of them are not designed to deal with semantic information and spatial analysis is hardly addressed (Tempfli, 1998).

Closely linked to the design of a 3-D representation, is the challenge of implementing the representation in a distributed environment. The direct acquisition of 3-D information for 3-D geo-information systems is very time consuming and expensive. 3-D GIS applications will be costly to the general populace, hence these resources ought to be shared so that maximum benefits are derived. Networks between different computer systems have been researched to facilitate this sharing (Dodge *et al.* 1998). The integration of GIS, VR and the Internet has been attempted in one way or the other, but there is still lack of a systematic investigation, which analyses different integration strategies is still lacking. In particular an appropriate approach for the efficient and effective integration needs to be addressed. Current Internet GIS deploys data and software to either the client-side or the server-side, causing a seriously unbalanced utilization of computing resources on both sides.

Related to distributed computing is the need for the user to interact with the models and related data. This is addressed by creating interactive web-based applications, in which the content of the

page is based on a user request. An early solution to this problem was the *Common Gateway Interface* (see Chapter 4); in which developers wrote programs to run requests on the server-side, and clients had to call the programs through the web server (Batty *et al.*1998). This solution has significant scalability problems; each new CGI request launches a new process on the server. If multiple users access the program concurrently, these processes consume all of the web server's available resources and the performance slows to a grind. With the advent of "plug-in" software, Java Server Pages Programming and Microsoft's Active Server Pages, the possibility exists for designing interactive systems which are free of "scalability" problems (Hall, 2003)

1.5 Research Objectives

The research objectives are summarised below as:

1. To design a Hybrid 3-D representation for urban feature modelling. The representation should allow for interactive spatial and textual analysis within a web-based browser client.

2. The second objective is directed towards the Distance Fields. The research will focus on the design and adaptation of distance field environment that would allow for interactive 3-D analysis and at the same time conserve space.

3. Lastly, the algorithms for interactive 3-D spatial analysis operations and a prototype will be developed.

1.6 Novel aspects of the research

The design and implementation of a hybrid 3-D representation for urban modelling that will allow for spatial analysis operations in a distributed environment using "off-the-shelf" Internet applications and tools, is new to Geomatics. The use of distance fields is slowly gaining momentum in medical imaging and its application into urban modelling is new. The research proposes algorithms for the implementation of a comprehensive set of 3-D spatial analysis operations (see Chapter 5), which is an achievement, as no known attempt has fulfilled this

objective. The research modifies the general distance field modelling map into, the single Distance Field (***sDF***), which is suitable for urban feature modelling (see Chapter 5).

1.7 Scope of the research

The 3-D hybrid representation is meant for urban applications. Texture mapping has been well studied in the literature (see Foley *et al.* 1996), hence it will not be covered in this research. The analysis of 3-D representations is restricted to the ones that have been implemented and those which have been publicised in known research work. Objects to be modelled are limited to those with definite (non-fuzzy) boundaries. The research will not cover issues relating to object data acquisition and instead it assumes that the object attribute and geometry data in form of point co-ordinates and attribute data are available.

1.8 Thesis Structure

The research is addressed within the following structure:

Chapter 1: Presents the introduction, problems in the area of 3-D modelling and the specific research objectives of this study. The research scope and novel aspects are also highlighted.

Chapter 2: This chapter gives a complete definition of a representation. Theoretical 3-D modelling terms and concepts necessary to this research are fully explained in this chapter. Set notation is employed in clarifying some concepts. A brief description of solid modelling is made.

Chapter 3: This chapter is an in-depth coverage of representations that include *boundary, constructive, spatial partitioning, sweep, primitive instancing, functional,* and *distance fields*. The chapter ends with a comprehensive comparison of the representations and gives the direction to be followed in creating "ideal" GIS representation.

Chapter 4: This chapter covers important modelling issues, which include: Distributed Computing environments and provides a detailed analysis of: client/server architecture, "fat" and "thin" clients and active web-page development tools

Chapter 5: Outlines the designs of the 3-D hybrid representation. Conceptual and Logical designs are provided. The chapter makes an exhaustive comparison of the boundary representations and adapts the sound theoretical principles from the different designs. Distance field modelling concepts are analysed and the single Distance Field (***sDF***) modelling technique is developed and adopted as the approach to use.

Chapter 6: Dwells on the physical design or prototype of the system. The framework for implementing the 3-D representation is proposed. Further, extensive efforts are made in identifying "off-the-shelf" tools to develop and implement the prototype.

Chapter 7: Provides an evaluation of the research findings.

Chapter 8: Documents the conclusions, recommendations and future research work.

Chapter 2
Three-Dimensional data structure (representation) Idealisation

This chapter seeks to give a basic description of a representation (data structure). The terms used and concepts defined will be used in the review of representations in chapter 3 and evaluation of the designed representation in chapter 7. Set theory concepts are used in the description. Characterisation of an ideal representation for 3-D Virtual City Modelling is done. The last part of the chapter looks at issues relating to internal structure of 3-D objects (solid modelling).

2.1 Representation

A representation or data structure is basically a way of organizing data representing physical entities so that it becomes suitable for developing and processing corresponding computer models.

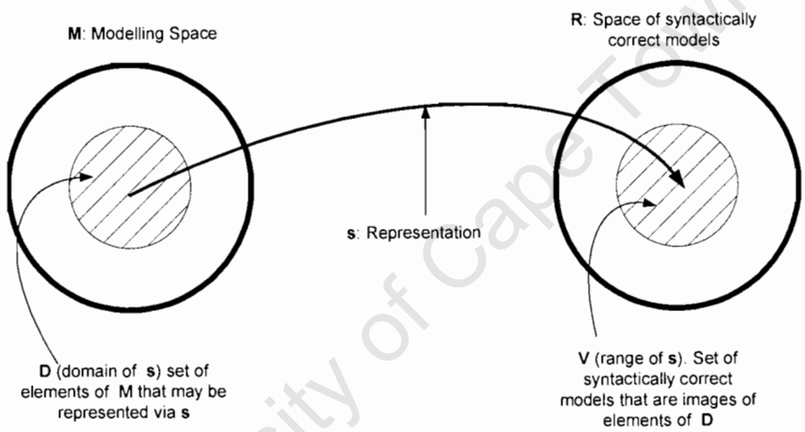


Figure 2.1: illustrating the definition of a representation

According to set theory, a representation is defined as a relation ***s*** that maps an entity of the physical world (which is within the domain, ***D,*** of the representation and in the modelling space, ***M***) to a computer model of the entity (within the range, ***V,*** of representation and contained in the space of syntactically correct models, ***R***), see Figure 2.1, above (Worboys, 1995).

$$relation\ s: M \longrightarrow R$$

M is a space whose elements are abstracted entities from reality and they are called mathematical models. In geometric computation, mathematical models usually are point sets in Euclidean space. The set of all *syntactically*[1] correct models (within ***R*** space) constitutes another model space. Models can be developed for various entities e.g. points, curves or solids.

2.2 Characterisation of a 3-D Representation

Research carried out so far has given rise to many representations, with different characteristics and uses. The research in this dissertation is basically concerned with 3-D Virtual City modelling and in this section, ideal characteristics for a representation will be given. It should be noted that it is generally regarded as exceedingly difficult to design a representation that meets all the requirements. The following subsections will provide a number of attributes that could be used to characterise representations:

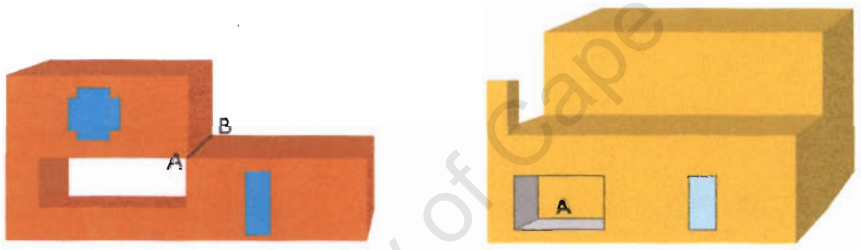


Diagram (a) **Diagram (b)**

Diagram (c)

Figure 2.2 The different complex models of the urban domain. Diagram (a) shows a complex building with a non-manifold property along edge *A-B*. Diagram (b) displays a complex structure with a hole, *A* in it. Diagram (c) shows the complex urban network which posses a real challenge in modelling.

Domain: The user of the system will certainly be interested in knowing the family of objects that can be modelled by a particular representation. This family is the domain. Requicha (1990) defines the domain ***D*** as a set of physical entities representable through the data structure (representation) ***s***. In an urban environment, this domain includes motorways,

[1] Syntactically correct model: This is a model, which obeys the rules of a particular representation (data structure).

bridges, drainage systems, buildings, overhead and underground utilities and modelling of pollution activities (see Figure 2.2, above).

Validity: A valid model (***r***) (note: ***r*** is a model found within the model space ***R***) is one, which is developed syntactically and semantically according to a particular known representation. Any model in the range ***V*** is said to be valid since it is both syntactically and semantically correct (i.e., it belongs to ***R*** and has corresponding elements in the domain ***D***). Model validity is of obvious importance in ensuring the integrity of a database, which should not contain models corresponding to objects that make no physical sense. A model is invalid if it is not abstracted from any physical entity (see Figure 2.3). In applications, invoking a geometric algorithm on an invalid model may produce a run-time error such as system crash however, this can be accounted by an algorithm that checks the validity before implementation.

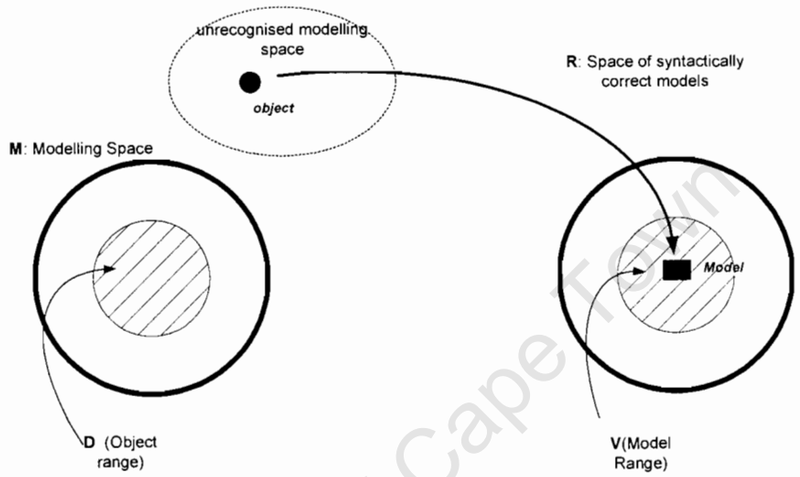


Figure 2.3: showing a invalid model

Unambiguity: An unambiguous representation should contain enough information to distinguish a single entity from all other entities in the modelling domain. Using the set theory, a model ***r*** in ***V*** is ambiguous or incomplete if it corresponds to more than one object (i.e. s^{-1} (r) maps to more than one element of ***D***). A representation is unambiguous if all of its valid models are unambiguous. An absence of ambiguity is crucial for the automatic computation of properties of the represented objects. Figure 2.4 below, shows two objects being mapped to give the same model. Hence there is ambiguity in the representation.

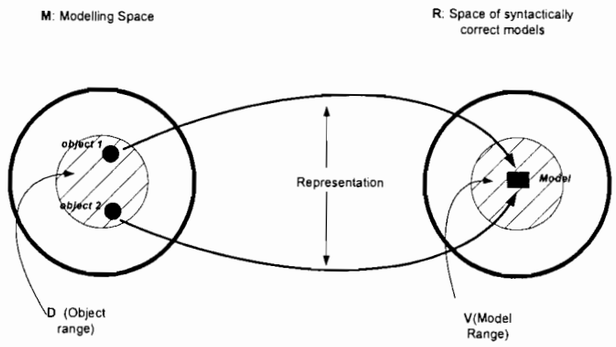


Figure 2.4. Illustrating ambiguity-objects 1 and 2 giving the same model.

Uniqueness. Does an object in the domain ***D***, have a unique model in *V*? A model is unique if its corresponding objects do not admit models other than ***r*** in the representation. A representation is unique if all of its valid models are unique. A 3-D object has non-unique models if it can be represented in several ways in the representation, Figure 2.5 below. An unambiguous and unique model establishes a one-to-one correspondence between its domain and range. Uniqueness greatly facilitates testing objects for equality. The following examples illustrate the need for equality –assessment algorithms.

- Repeated models may be culled from a database only if it can be determined that the two models (from the same representation) correspond to the same object.
- Automatic planning algorithms, which search for alternatives, may loop indefinitely if they cannot recognize previously encountered situations. In this case, the algorithms will fail to match an object to a specific model because of lack of uniqueness.
- Correct programs for numerically controlled machine tools must produce objects that equal those specified by the product designers. In this case, the programs may give rise to more than one possible model if there is lack of uniqueness.

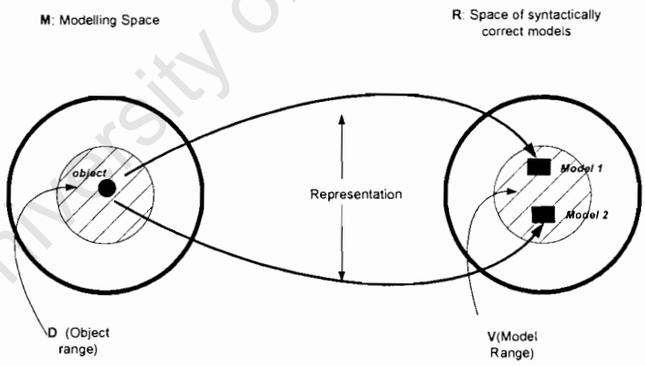


Figure 2.5 Lack of Uniqueness

Conciseness: How large are the models? In 3-D urban modelling, the objects of interest are complex structures and they consume Gigabytes of computer memory. The challenge for a representation is to reduce complex structures into concise models. Conciseness refers to the "size" of models for a particular representation. Concise models are convenient to store and to transmit over data links, and contain relatively little redundant data.

Ease of Construction: How hard is it to construct a valid model? Models in verbose representations with complex validity conditions are difficult to construct. Concise representations are easier to create than verbose ones, because conciseness implies that fewer data need be specified and that the individual data items are largely independent.

Modelling systems based on verbose representations usually contain powerful input subsystems to help users with the creation of models. Such modelling systems should possess automatic validity, ensuring mechanisms to relieve users of the burden of ensuring the validity of models.

Analysis Operations: If 3-D modelling systems are to become acceptable to traditional 2-D system users, they ought to offer more operations and functionalities than those currently enjoyed in 2-D systems. Because 2-D systems lack the third dimension, they cannot provide the user with 3-D spatial analysis. The main approaches in geo-information systems used in encoding spatial relationships are metric and topology. The metric is a pure computational approach, based on comparison of numerical values related to the location of objects in the Euclidean space. For example, the spatial relationship between a house and a parcel (e.g. inside, outside, to the south) can be clarified by metric operation and topology: *point-in polygon* performed for each point constituting the footprint of the building. Since metric is built on the notion of the distance function, the approach is computationally expensive, especially when considered in 3-D. Table 2.1 below, provides a number of operations that are of interest to this research work.

Topology: Topology is fundamental to much of the analysis and many of the queries posed in 3-D systems. Hence the model will be expected to capture the topology as accurately as possible.

Geometry gives the location and shape in space as well as dimensions of an object, while topology allows us to take into account the various relationships between the objects in space (adjacency, inclusion, overlapping). Note that topology is a complement of geometry and that,

often can be deduced from it, but at the expense of additional computation. This possibility implies that the explicit storage of topology is in effect redundant, but it is useful for the efficiency of some processes.

The first consideration in topology is spatial semantics (i.e. description of a feature in space relative to others e.g. adjacency, containment etc.), Brisson (1989). These spatial semantics is more relevant for users than geometric co-ordinates. Indeed, in common language, we designate an object site by giving its location among other objects (e.g. ***X*** is located in room-***A*** which is adjacent to room ***B***). Therefore, topological semantics will allow a user, to query data according to topological criteria.

Topology can be useful for further tasks such as:

- Speeding up processing which uses properties of the connection between objects
- Reducing space needed for storage by sharing common data (e.g. two contiguous objects sharing a frontier).

Query/Operation	Description	Relevance
Neighbourhood operation	This includes answering queries relating to – inside, overlap, covered by, meet, disjoint, contains, inside. The operation uses the concepts of binary topology.	This is by far the most common operation on which GIS analyses are based.
Retrieve/Select	According to user defined conditions, Results will be displayed for browsing	The queries are based on the semantics of features. This is quite popular with ordinary users of Geometric Modelling Systems because they are not complicated at all.
Update/Delete Operations	This is all about introducing new features and deleting some from the spatial database.	Experts are maintaining most 3-D Modelling Systems, and interactive updates are still to be fully implemented. The ordinary user of the system rarely updates or deletes data. So the ranking for this operation is quite high.
Interactive Navigation Operation	This is implementable in Virtual Reality Systems. It allows the user to navigate in a VR set-up and make queries at the same time.	Most VR systems have been developed for passive navigation. The user is interested in making queries on the fly. With the move to replace 2D maps with 3-D cities, this operation is of paramount importance.
Buffering	This involves the generation of spaces around features of interest using set criteria.	This is important to expert users of the system, especially for planning purposes.
Network analysis	Involves analysis operations carried on linear features-connectivity, length, shortest path, gradient etc.	This is important for a specialised sector of the expert user community- cabling, telecommunications, transportation etc.
Visibility	This involves the need to establish inter-visibility between features of interest.	This is important to utilities organisations, telecommunications, transportation etc.

Table 2.1. Common spatial operations

Progressive transmission. As stated earlier, 3-D urban data is quite voluminous and very expensive to acquire. Organisations share data over the net to avoid its duplication. Users benefit from this use of remotely located the data. For this to be effective the data needs to be accessible and well-structured to allow for distribution over networks.

Progressive transmission entails that data is not transported all at once. The 3-D representation should support processing algorithms which allow data at different Levels of Detail (LODs) (De Floriani et al. 1999). For this to be possible, the data should be stored in hierarchical structures.

Visualisation: The user interacts with 3-D objects using the appropriate visualisation tools and environments. Effective visualisation is ensured by data representations, which allow easy parsing and rendering. The representation must readily support high quality graphics.

2.3 Solid Modelling

In some instances the internal structure of an object is of particular interest, for example in underground of mineral ore bodies. The representation has to exhibit the following characteristics if it is to fully abstract the internal structures of objects (Elmqvist, 2001):

Rigidity: A solid defined in $\boldsymbol{E}^3$E3 should have an invariant shape which is independent of the of the solid's location or and orientation (Foley *et al.*1996).

Finiteness: A physical object should have a finite extent (as opposed to a fuzzy extent). To ensure finiteness all we need to do is require that our data sets be bounded, i.e. a boundary has to be established.

Solidity: A model for a solid should be homogeneously 3-D, without dangling faces or edges. This requirement can be met by regular *r-sets*[2] in 3-D Euclidean space (Requicha *et al.*, 1985).

Closure under Boolean operation: Boolean operations applied to solids should produce other solids. This has two important advantages: first, the results of a Boolean operation can be used as input to other Booleans, and a solid model can therefore be incrementally

[2] r-sets: set of all the 3-D objects that are manifold, and can produce regular solids under Boolean operations.

constructed by successive Boolean operations. Second, subtractive and additive processes are guaranteed to produce solids, e.g. alterations to buildings or infrastructure.

Boundary determinism: solid should be modelled by a set that is unambiguously defined by its boundary. Without boundary determinism we will not be able to use *B-reps* (see Chapter 3), which are one of the most popular schemes for representing solids.

2.4 Computational Complexity

In the design of data structures, it is generally important to regard the efficiency of a representation. Efficiency covers a lot of issues, but in this research, efficiency will be directed at computer memory and algorithmic time resource.

It is however important to differentiate between the following:

- Performance: how much time or memory is actually used
- Complexity: how do the resource requirements scale i.e. what happens as the size of the problem solved gets larger. This is also referred to as asymptotic complexity measure.

2.4.1 Big "O" Notation

Big "O" notation is used to express computational complexity (see Rohlf *et al.*1994). For a problem size of N:

- A constant-time is "order 1": $O(1)$
- A linear-time method is "order N": $O(N)$
- A quadratic-time method is "order N squared": $O(N^2)$

Formal definition: $f(n) = O(g(n))$ means there are positive constants c and k, such that $0 \le f(n) \le c(g(n))$ for all $n \ge k$ (see Rohlf *et al.*1994). The values of c and k must be fixed for the function f and must not depend on n.

Note that the big-O expressions do not have constants or lower order terms, This is because, when N gets large enough, constants and lower order terms do not matter (a

constant time method will be faster than a linear time method, which will be faster than a quadratic time method, see table 2.2 below) (Finkel *et al.*1988)

Big Oh	Comment
O(1)	Very fast, constant time, independent of input
O(log n)	Fast (e.g. binary search)
O(n)	Linear time (e.g. linear search)
O(nlogn)	(e.g. optimal sorting algorithms)
$O(2^n)$	Slow, exponential.

Table 2.2 : summary of the Big "O" value range

Chapter 3
Analysis of 3-D representations

This chapter gives a detailed analysis of the existing data representation categories in 3-D modelling. The analysis is based on the attributes defined in chapter 2. The last section of this chapter gives a comparison of the representations and proposes the direction to be taken by this research work.

3.1 Boundary representations (B-reps)

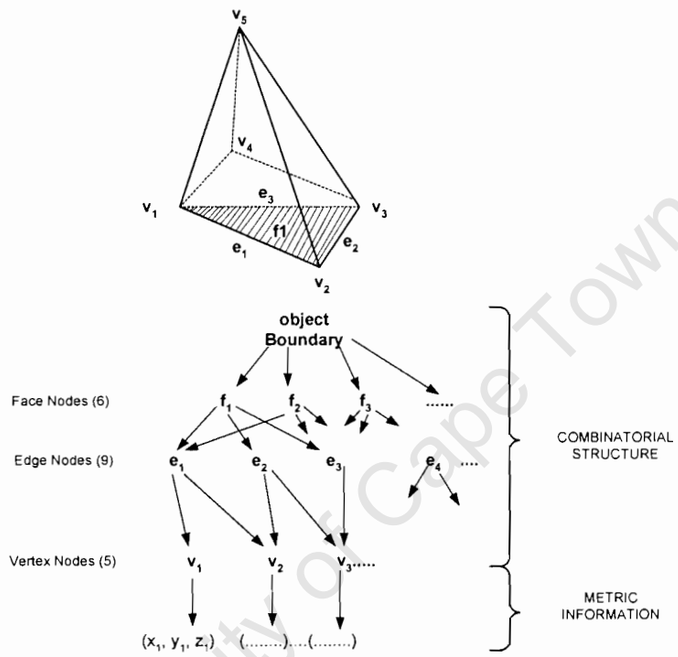


Figure 3.1. A boundary representation for a rectangular pyramid [adapted from Requicha, 1980]

The boundary data structure is one of the most popular representations. This is mainly because many of the questions in virtual city modelling, we seek answers to, relate to the surface of a physical object (Stouffs *et al.* n.d.). A boundary representation models a solid by representing its boundaries as a collection of two-dimensional boundary surfaces or faces. The boundary in such is usually segmented into a finite number of bounded subsets, which are known as faces or patches. The faces, in turn, are bounded by edges and vertices, which

serve as building blocks in such a representation. The adjacency relationships of faces, edges and vertices are maintained as topological information.

Figure 3.1 provides an example of a boundary representation for a rectangular pyramid. Essentially, a B-rep is a graph structure with nodes corresponding to faces, edges and vertices in a cell decomposition of the solid's boundary. Links between the nodes express connectivity information. Observe that the top levels of the graph in the Figure contain only connectivity information. This constitutes the combinatorial structure of the representation. The vertex co-ordinates are the metric information associated with the representation. In the geometric modelling paradigm, the combinatorial structure is often called the topology, and the metric information, the geometry.

In boundary representations, faces should satisfy the following characteristics (Requicha, 1980):

a) A face of an object is a subset of the object's topological boundary
b) The union of all of the faces of an object equals the object's boundary
c) Each face is a subset of only one primitive surface instance
d) A face is homogeneously 2-D; it has no dangling vertices or edges (for 2D *manifold* conditions).
e) Faces are quasi-disjoint, i.e. they only meet at edges or vertices

Edges also must be defined precisely. Typical B-reps use edges that satisfy the following conditions:

a) An edge is a subset of a face's boundary or line feature.
b) The union of all edges associated with a face equals the face's boundary
c) Each edge is a subset of the intersection of two primitive surface instances
d) Edges are quasi-disjoint, meet only at vertices

In the following section, a detailed account of B-rep representations will be given. The representations will be analysed under two broad classes: *manifold* and *non-manifold* representations.

3.1.1 Manifold representations

Many B-reps support only solids whose boundaries are 2-*manifolds* (Foley *et al.*1991). A point is considered to be on a 2-manifold when an arbitrary small neighbourhood of points

around it can be considered topologically the same as a disk in the plane (Blinn, 1982). This means that there is a continuous one-to-one correspondence between the neighbourhood of points and the disk (see Figure 3.2, below).

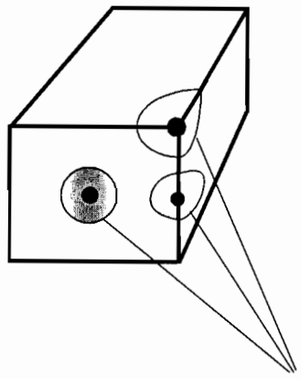


Figure 3.2. 2-manifold illustration. Each point, shown as a black dot, has a neighbourhood of surrounding points that is a topological disk, as shown in grey.

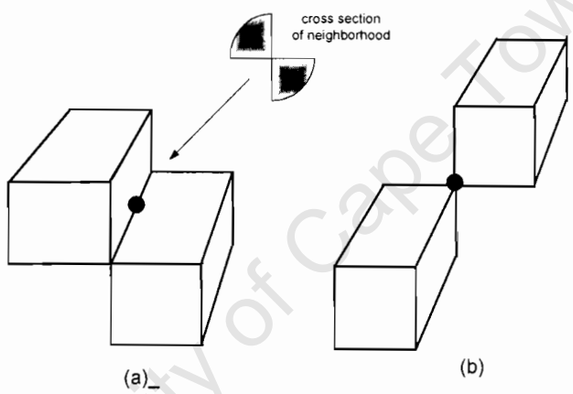


Figure 3.3. Non-manifold illustration. The boundaries of two cubes glued at an edge or at a vertex are not 2-manifolds.

In contrast, Figure 3.3 above, shows two polyhedral surfaces that are not manifolds. The object on the left (Figure 3.3(a)) fails to be a manifold because it has an edge that is shared by more than two faces. The neighbourhoods of points on the common edge are made out of four half disks that cannot be flattened into a single disk without gluing. The object on the right (Figure 3.3(b)) has a vertex that is shared by two cones of faces. Its neighbourhoods can be

deformed into two disks, but these have to be glued to obtain a single disk. A surface that crosses itself has at least one edge of self-intersection, and therefore is analogous to the two cubes glued along an edge in Figure 3.3. It follows that self-intersecting surfaces are not 2-manifolds. It is also clear from this Figure that the union of two 2-manifolds (the cube boundaries) may produce objects that are not manifolds.

Winged-Edge representation: It is widely regarded that most of the **B-rep**s that have evolved over the years have done so from the fundamentals of the winged edge data representation (Kettner,1999). The innumerable variations proposed include: the Hybrid edge (Kalay, 1989), Bridge edge (Yamaguchi *et al.*, 1985), half edge (Mäntylä, 1988), Split Edge (Eastman, 1982), Doubly connected or DCEL (Preparata, 1985), Quad edge (Guibas *et al.*, 1985), Radial edge (Weiler, 1985). The quest was mainly towards the reduction of redundancies in topological information capture and modelling of non-manifold solids.

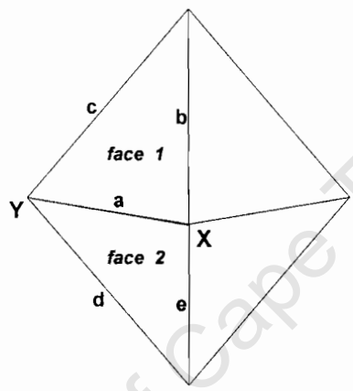


Figure 3.4:Polyhedron "marked" for winged-edge modelling

The winged edge representation derives the name from its graphical appearance (see Figure 3.4) (Baumgart, 1974). In this representation, a solid consists of faces, a list of edges and a list of vertices. The above diagram (Figure 3.4), shows a polyhedron with a sample of vertices, edges and faces marked for the purposes of winged edge modelling. Analysing edge ***a*** = **XY**: this edge has two incident vertices ***X*** and ***Y***, and two incident faces, ***face 1*** and ***face 2.*** A face is a polygon surrounded by edges. For example, face ***1*** has its edges ***a***, ***c*** and ***b***, and ***face 2*** has its edges ***a***, ***e*** and ***d***. The ordering is clockwise as viewed from outside of the solid. If the direction of the edge is from ***X*** to ***Y***, ***face 1*** and ***face 2*** are on the right and left side of edge ***a***, respectively. To capture the ordering of edges correctly, four more pieces of information. Since edge ***a*** is traversed once when traversing ***face 1*** and traversed a second time when traversing ***face 2***, it is used twice in different directions. For example, when traversing the

edges of ***face 1***, the predecessor and successor of edge ***a*** are edge ***b*** and edge ***c***, and when traversing the edges of ***face 2***, the predecessor and successor of edge ***a*** are edge ***d*** and edge ***e***. Although there are four edges incident to vertex ***X*** (see, Figure 3.4), only three of them are used when finding faces incident to edge ***a***. Therefore, for each edge, the following information is important:

1. vertices of this edge,
2. its left and right faces,
3. the predecessor and successor of this edge when traversing its left face, and
4. the predecessor and successor of this edge when traversing its right face.

Each entry in the edge table, Table 3.1 below, contains the information mentioned above: edge name, start vertex and end vertex, left face and right face, the predecessor and successor edges when traversing its left face, and the predecessor and successor edges when traversing its right face. Note that clockwise ordering (viewing from outside of the polyhedron) is used for traverse. Note also that the direction of edge ***a*** is from ***X*** to ***Y***. If the direction is changed as from ***Y*** to ***X***, all entries but the first one in Table 3.1 must be changed accordingly.

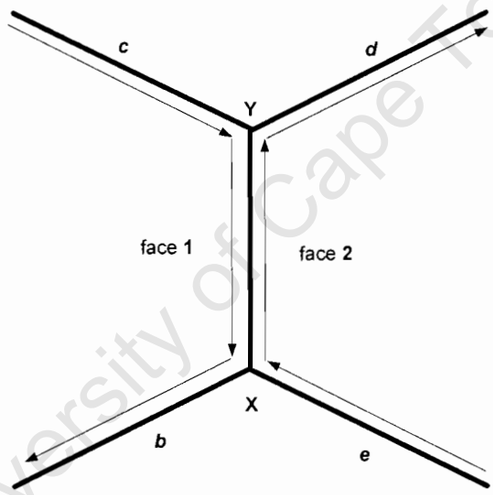


Figure 3.5: Edge and Face orientations for a cube

Edge	Vertices		Faces		Left Traverse		Right Traverse	
Name	Start	End	Left	Right	Predecessor	Successor	Predecessor	Successor

Table 3.1 Winged-edge topology table

Table 3.1. Shows the information for the entry of edge ***a***. The four edges ***b***, ***c***, ***d*** and ***e*** are the *wings* of edge ***a*** and hence edge ***a*** is " winged."

3.1.2 Non-Manifold representations

As mentioned earlier, most conventional modelling revolves around manifold representations, which have their own restrictions in failing to capture information for non-manifold structures. Kevin Weiler (1985) proposed a radial Edge data structure for representation of non-manifold topologies. This was the first complete non-manifold boundary representation. Yamaguchi *et al.* (1990) developed a non-manifold representation based on neighbourhood relationships. Gursoz *et al.*(1990) introduced a modified version of Weiler's Radial edged representation.

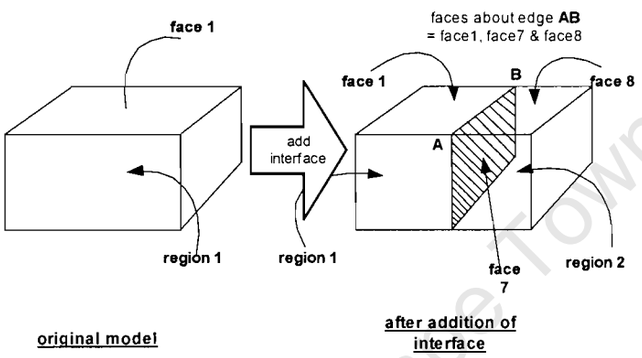


Figure 3.6. Radial-edge representation for a non-manifold condition.

Figure 3.6 above, illustrates the mechanism by which the radial-edge structure represents non-manifold conditions. Suppose one was to represent a solid hexahedral structure composed of two separate materials. Such a model could be constructed by starting with a single solid, and then add an interior surface (topologically equivalent to a face) to split this box into two regions, each of which could then be assigned appropriate material property attributes. The interface addition results in the splitting of one region, four faces, and four edges: the addition of four edges and four vertices, as well as the addition of the interface (face) itself. A non-manifold condition occurring along each of these added edges for example, edge ***A-B*** in Figure 3.6 above. The radial-edge structure stores the adjacency information of faces about this edge. One can loop radially about the edge **A-B** and extract the list of adjacent faces: [face8, face7 and face1].

3.1.3 Properties of B-reps:

This section summaries the B-rep properties:

Domain: Boundary representations are potentially capable of covering domains as varied as those covered by all other structures. The different representations range from modelling shapes effectively to accommodating non-manifold objects. In trying to capture the curvature of objects, splines are often used to replace edges. This can, however, cause numerical problems, because points on the edge do not necessarily lie in the surfaces whose intersection creates the edge. Approximate representations in the (u, v) parametric spaces of the surfaces are even more pernicious, because each edge is represented twice, and the representations may not coincide exactly (Requicha, 1980). Since B-reps are concerned with surfaces of object, they do not model the internal structures.

Unambiguity: Ambiguous modelling situations are not common in B-reps. Two objects in space cannot share the same entity in the model domain. This can be attributed to the fact that it is highly unlikely to have two different objects with similar boundary subdivisions. Because objects will have different surface decompositions, they will have different faces and as a result, different models. In spatial systems, one would not expect two objects to occupy the same position within the model domain (note: the systems will have a constraint imposed on them which will ensure non-existence of spatial coincidence between models).

Uniqueness: Faces generated are generally neither unique nor complete. That is, an object might have two or more models in the model-domain, a case of one-to-many relationship (Requicha, 1980). The lack of uniqueness in B-reps arises from the fact that curved face edges can be approximated (see Figure 3.7, below). Depending on the how the approximating parameters vary, there is a possibility of getting many different models from the same object.

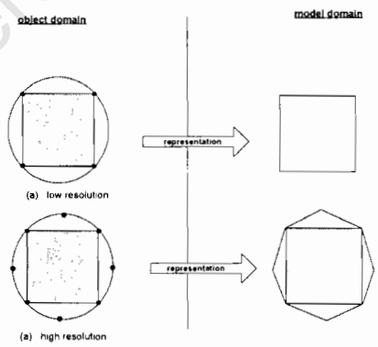


Figure 3.7. Uniqueness in B-reps

Validity: The validity of B-reps raises interesting issues. To study validity, one proceeds "bottom-up" by investigating the necessary conditions for vertex nodes, to represent points in the Euclidian space (object domain), for edge sub graphs (i.e., edge nodes plus their corresponding vertex nodes) to represent line segments, and so on.

Two types of conditions for validity can be distinguished as combinatorial and, metric (Agoston,1976).The following are the combinatorial conditions:

(1) Each face must have at least three edges
(2) Each edge must have precisely two vertices
(3) Each edge must belong to at most 2 faces (manifold shapes) or at least 2 faces (non-manifold).
(4) Each vertex in a face must belong to precisely to two of the face's edges.

Conditions (1) and (2) are obvious. Conditions (3) and (4) ensure, respectively, that the surface "closes" and that each face's edges form "loops". Algorithms to test the combinatorial conditions above can easily be developed (Eastman *et al.*, 1979).

Combinatorial conditions do not suffice to ensure validity, the following metric conditions must also be satisfied:

(a) Each three-tuple of vertex co-ordinates must represent a distinct point in Euclidian space (i.e. object domain).
(b) Edges must either be disjoint or intersect at a common vertex.
(c) Faces must either be disjoint or intersect at a common edge or vertex.

It can be concluded that, validity checking for B-reps is not computationally attractive. Most geometric systems attempt to embed the required validity conditions in the algorithms used to construct the models, instead of testing representational validity after the B-reps models are built (Requicha, 1980).

Conciseness: Boundary representations are described as verbose (Requicha, 1980). Figure 3.1 illustrates this verbosity by indicating the varying constructs (vertices, edges, faces and bodies) that add up to a 3-D model. The validity conditions requirement, adds on to the verbosity of B-reps.

Ease of Construction: This sheer volume of data required and the delicate validity conditions, make it difficult for humans to construct, without computer assistance, boundary representations of even moderately complex objects. Because of this verbosity, almost all known geometric modelling systems attempt to provide some modelling assistance (Reddy *et al.* 1978).

Efficiency: The main virtue of B-reps lies in the ready availability of models for faces, edges and the relations between them; this is important for generating line drawings and graphic displays and for supporting graphical interaction. It is, thus not surprising that B-reps have been extensively used in computer graphics.

Many B-reps contain additional information that is redundant but important for algorithm efficiency. For example, face normals may be stored, or edges may have back pointers to faces, which share them. Additional links are used to facilitate graph traversal. The winged edge representation is an example of such a heavily linked B-rep and it is the intellectual root of many of the data structures in use today.

Regardless of the availability of topological information, studies conducted by Ala (1991) and Woo (1985) indicated that the efficiency of the winged edged data and its derivatives remain an area of concern. This however, is an issue for further research in this thesis (see chapter 5).

Topology: Because B-reps have the combinatorial component in their general structure, hence topology is captured as part of the representation. The topological properties of connectedness and adjacency are readily provided. Because the B-reps have been employed extensively in computer graphics, known algorithms to compute and capture topological properties of objects have evolved. (Preiss, 1979).

Analysis operations: The B-reps can be used effectively in 3-D geometric systems for neighbourhood analysis operations. Point Membership Classification (PMC) is a big challenge (see chapter 5). The implementation of PMC involves the concept of *ray-casting* (Worboys, 1994).The computational complexity of the ray-casting process grows with the number of faces in a boundary surface. Consider for example, the case of a simple sphere approximated with 288 flat faces. PMC against its surface requires at least 282 line-plane intersection calculations in 3-D. It can be concluded that the PMC process in B-reps is a big operation (Rossignac *et a.l* 1989). Like the PMC operations, buffering in 3-D poses similar

challenges and is yet to be implemented. Merging solids through Boolean operations is among the most difficult of B-rep operations (Worboys, 1994).

3.2 Constructive Representations

Constructive representations can be described as a means of capturing a construction process which defines a solid This process is a sequence of operations that instantiates or combines modelling primitives or the results of previous constructions (Rossignac *et al.*, 1992).

3.2.1 Constructive Solid Geometry (CSG)

Constructive Solid Geometry (CSG) is the most popular constructive representation. It represents the solid object as combinations or constructions of solid components (primitives) using the *regularized set operators* and linear transformations: rotation, translation and scaling (Watt, 1993). The CSG primitives are simple objects such as cubes, boxes, tetrahedrons or quadratic pyramids. The primitives can be instantiated by the user in his design. They are considered bounded point sets in 3-D space, and can easily be combined using Boolean set operations (union, intersection and difference) in order to represent more complex objects. The most natural way to represent a CSG model is the *CSG tree* (Figure 3.8, right). The leaves of the CSG tree are the primitives, and the nodes are marked either as a Boolean set operator or with a rigid motion. Thus, in the CSG tree, the history of generation of the solid is stored. The solid itself corresponds to the upper node of the CSG tree (Mäntylä, 1988).

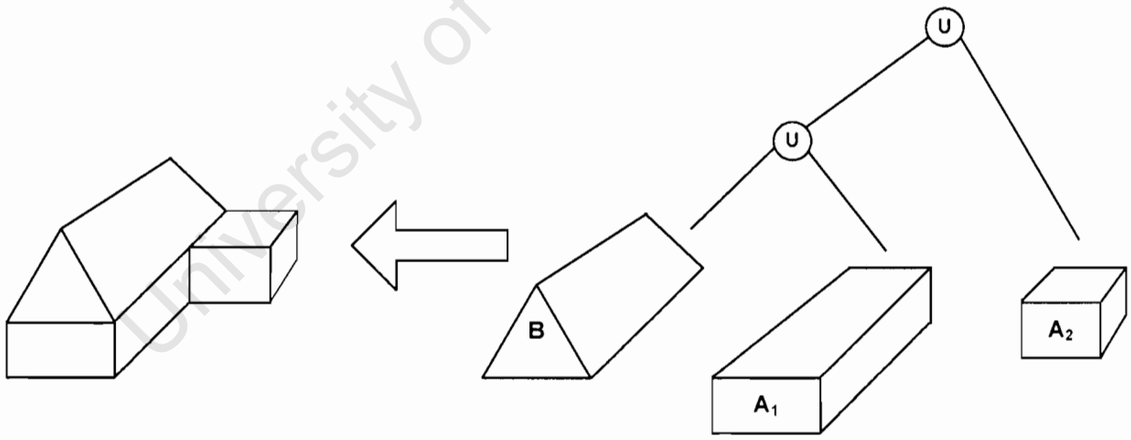


Figure 3.8: The CSG model (left) is represented by the CSG tree (right) consisting of three primitives, connected by Boolean union operation (U).

3.2.2 Properties of CSG representation

Domain: CSG is not as flexible as a boundary representation: the applicability of CSG depends on the primitive set available (Müller, 1998). If an inappropriate set of primitives is offered, object modelling using these primitives will become difficult and hence the domain will be restricted to the derivates of the primitives available.

Unambiguity: The potential for ambiguity does not exist with CSGs. Each CSG tree is developed for a particular object creating a one-to-one mapping (Foley *et al.* 1991)

Uniqueness The lack of uniqueness within the CSGs is mainly due to the fact that different primitives can be combined resulting in a varied number of models for the same object (Müller, 1998).

Validity: CSG trees are always guaranteed to represent valid objects. It is important to realise that the "guaranteed validity" of CSG schemes based on solid primitives applies only to schemes in which combinatorial operators are regularised set operator (i.e. operators which will produce shapes which are bounded and without dangling edges)(Foley *et al.* 1991). To validate CSG's, only simple syntactic checks of the CSG-tree are needed

Conciseness: CSG schemes whose primitives are well matched to the domain of objects to be represented are very concise (Requicha, 1980). For example, the object in Figure 3.8, would require a few instances of only two primitives: A_1 and A_2 are instances of the same primitive and B is an instance of a pyramid.

Ease of Construction: Models that correspond to the domain of primitives can be constructed easily .The ability to edit models by deleting, adding, replacing, and modifying sub-trees, has made CSG one of the dominant solid modelling representations (Dong, 1991). CSG is commonly used in solid modelling in CAD/CAM because object creation can be completed with a simple modelling language (Raper, 1989).

Efficiency in the context of Applications: CSG representations are convenient for implementing application algorithms that are simpler and often more robust than their B-rep counterparts (Rossignac, 1986). Many solid modelling algorithms work directly on CSG

representations through a divide-and conquer strategy, where results computed on the leaves are transformed and combined up the tree according to operations associated with the intermediate nodes (Landford *et al.*,1994). PMC algorithms are very simple to implement in CSG. The point is classified with respect to primitive solids, and classification results are recursively passed up the CSG tree (Requicha *et al.*, 1985). Efficient algorithms do exist which record changes in Boolean operations and transformations, since they involve only tree-node rebuilding.

Topology: CSG representations do not explicitly carry information on connectivity or even the existence of the corresponding solid. These topological questions are best addressed through some form of boundary evaluation, where a whole or partial B-rep is derived algorithmically from the CSG model (Rossignac *et al.* 1996) A lack of topological information implies that exhaustive computations have to be carried out every now and then (Naylor, 1990).

Analysis operations: Some tasks such as the classification of a point as being inside or outside the solid are very simple to solve using CSG (Tilove, 1980). The standard point membership classification algorithm for CSG representations traverses the CSG tree by recursive descent. When the recursion reaches the primitives, the point to be classified is tested against them. Once the point/primitive classification results are known, they are combined through the original Boolean expression that defines the CSG graph. The process is simple if the primitives occupy natural positions (i.e. when the primitive's axes are aligned with the object's principal axes). More complex primitives may involve intersecting sets defined by inequalities, or more general point containment tests.

However, for certain operations in visualization, a boundary representation must be derived from a CSG model. This operation, called boundary evaluation, is rather complicated. First, the CSG primitives have to be converted to boundary models, and then these models have to be combined using the Boolean set operations, which, turns out to be very involved (Foley *et al.*, 1991). Note that the conversion cannot be inverted: whereas in CSG trees, the history of generation of a solid model is stored, this information is lost during the conversion to B-rep, and there is no general method available to reconstruct CSG trees from B-rep.

3.3 Spatial-Partitioning representations

With spatial partitioning representations, the idea is to partition space into regions, called cells, and enumerate those cells, which are filled with material, and therefore constitute the object being represented.

The entire 3-D space, or sometimes just the set that corresponds to the solid, is partitioned into non-overlapping 3-D regions called cells. Solids may be represented either exactly or approximately by a variety of space decomposition representations.

3.3.1 Spatial occupancy enumeration

Spatial occupancy enumeration represents space as an array of adjacent cells called voxels, whose cells are occupied by the object being represented. Typically these data structures are stored as a one-dimensional array of elements, each of which is defined by the *x, y* and *z* co-ordinates of the voxel centroid. Each element can then store a set of attributes or just a Boolean value to indicate whether it is totally inside or outside the object. The interior volume is the union of the cells that are totally inside the body. The boundary of the object is approximated by staircase boundaries of the cells. Figure 3.9 below, provides an example of a church complex modelled through spatial occupancy enumeration.

In order to address the demanding storage requirements of the spatial occupancy enumeration, a data storage technique, the Octree were developed (Samet, 1990). Octrees are in turn derived from quadtrees, a 2-D format used to encode images.

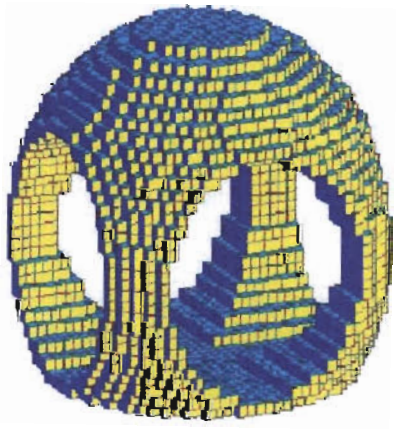


Figure 3.9. Spatial occupancy enumeration being employed in modelling a church complex.

3.3.1.1 Octrees

Octrees are often used for encoding three-dimensional objects because they enable large object blobs to be represented, addressed, and processed with the same space and time complexities as those of a single volume element or voxel (Kavouras *et al.*, 1988), see Figures 3.10 and 3.11, below. The octree is a rooted, directed graph generated by recursive subdivision of an initial $2^N \times 2^N \times 2^N$ raster into nodes or octants (where N is the root node level. Level $N = 0$ corresponds to the root cube before subdivision: $N \geq 0$). If all voxels forming a node have the same attribute, that node becomes a leaf with that attribute and recursive subdivision is halted, otherwise the octant is subdivided into eight sub-octants, and the process repeats recursively to a limit of N times. Extensive coverage on octrees can be found in Eastman *et al.* (1979).

For example, Figure 3.10 shows a simple object, along with its classical octree, and parenthesised linear representations. If a tree representation is used, a total of 17 nodes is required, each of them having eight pointers. However, its linear encoding requires only an amount of 34 bits, since the type of every node (*black*, *white* or *grey*) can be stored in two bits (Brunet *et al.*, 1985), and the parenthesis characters are not themselves needed, since they are used just for clarity purposes. The Octant numbering starts from a root node ***C***. Childs to the root node are shown in brackets (***W*** and ***B***).

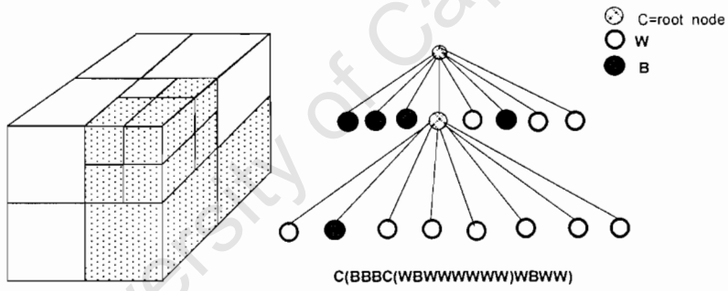


Figure 3.10: A simple object (left). Its classical octree (with max depth = 2), and parenthesised linear representation. (centre): Octant numbering.(right)

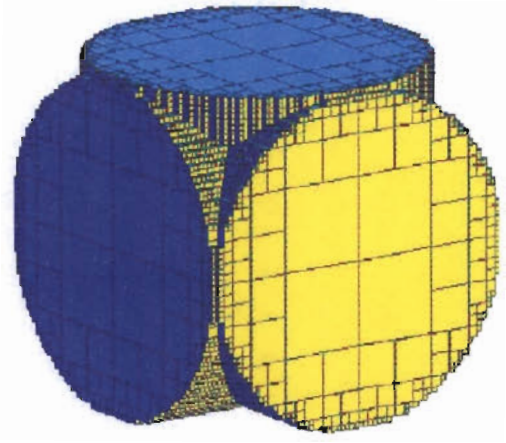


Figure 3.11. Octree concept in modelling a building. Voxels are shown as getting smaller as one moves to the edges of the object.

3.3.2 Cell Decomposition

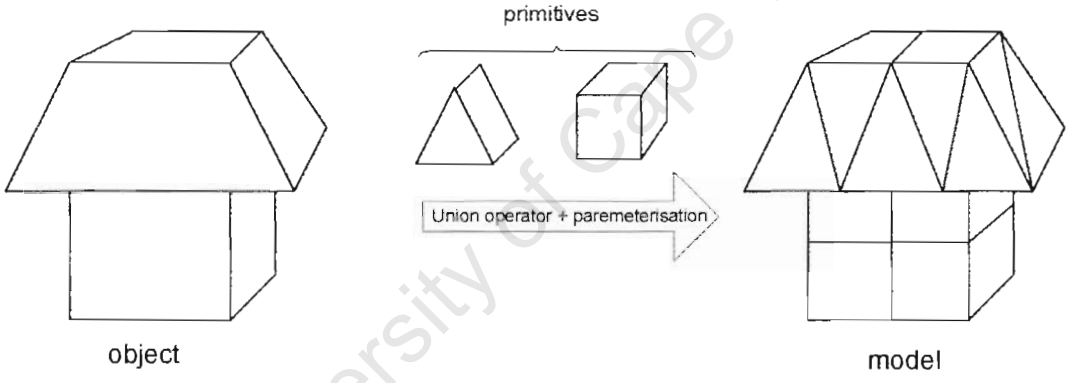


Figure 3.11: Cell decomposition. Object (left) being mapped to model (left) using the primitives and operator (centre) combined used to construct the model(right).

Complex objects are best analysed in terms of a decomposition of space cells of various dimensions (volumes, faces, edges and points) and in terms of their support for selecting arbitrary combinations of such cells. Cell decomposition is analogous to triangulating a solid, but triangles and tetrahedra are replaced by 2-cells and 3-cells respectively (i.e. by solids and 2-D faces of varying shapes). Unlike the spatial occupancy enumeration in which all the cells lie in a fixed regular grid, cell-decomposition cells lie along axes which are neither parallel nor orthogonal to one another. Cell decomposition differs from primitive instancing in that more complex objects are composed from simple, primitive ones in a bottom-up fashion by "gluing" them together. The primitives are combined or glued together through the union

operation in which they are not allowed to intersect (Requicha, 1980). Figure 3.11 above, gives an example of this particular representation.

3.3.3 Properties of Spatial Partitioning representations:

Domain: The domain of this representation technique is, theoretically infinite. Unlike B-reps which model the boundary of structures, spatial partitioning representations capture the internal components as well (Blanchin and Chiles, 1991).If the cells are cubes, then the only objects that can be represented exactly are those whose faces are parallel to the cube sides and whose vertices fall exactly on the grid (Foley *et al.*, 1991). More detailed and accurate models can only be achieved by increasing the resolution of the voxels.

Unambiguity: Like CSGs, the spatial occupancy enumeration representations generate an a hierarchical tree (i.e. an octree) for every model. Because each tree is specific between an object and a model, hence there is no ambiguity. The same can be said of cell decompositions (Preiss,1979).

Uniqueness: Cell decompositions lack uniqueness as a result of the fact that different sets of primitives can be combined to model the same object resulting in a number of different models being generated. Spatial Occupancy enumeration reps develop unique models because they have specific ways of generating them (Preiss,1979).

Validity: Cell decompositions are generally difficult to validate since each pair of cells must be tested for intersection. Because of different cell types and sizes, the validation process is computationally expensive as the intersecting planes are oriented in different directions implying a multitude of plane equations (Agin,1976). On the other hand, spatial occupancy enumeration representations can be easily validated because of the inherent regular grid structure. It is easy to check for cell alignments and overlaps.

Conciseness: Spatial partitions are potentially quite verbose. The degree of verbosity depends on the complexity of an object being modelled. A regular object being modelled by spatial occupancy enumeration will have a less verbose structure compared to a complex one in which there is a lot variations in cell resolution in order to map the intriguing detail.

Ease of construction: Cell decompositions are not easy to create and algorithms for constructing them are an area of current research (Bernadini *et al.* 2000). On the other hand, models can easily be created using the spatial enumeration approach.

Efficiency in the context of applications: Spatial enumeration is often used in biomedical applications to represent volumetric data. Many medical scans, such as CT (Computed Tomography), MRI (Magnetic Resonance Imaging) and PET (Positron Emission Tomography), yield data in 2-D slices, which can be assembled by a computer and rendered as voxel images. These 3-D structures are easier for surgeons to comprehend than a set of individual 2D sheets.

The spatial occupancy enumeration representation has distinct advantages: (Frank *et al.*, 1989): space is uniquely defined; cells are spatially indexed implicitly through the use of regular grids (cubic). This means that each cell is easily addressable thus allowing efficient spatial searching.

The most significant disadvantage of using voxels, however, is the amount of memory required to store them, which results in a high space complexity , for example, up to n^3 occupied cells are needed to represent an object at a resolution of n voxels (Frank *et al.*, 1989).

Topology. Topological information is not easily obtained from spatial partition. The high combinatorial and geometric complexity is a limiting factor since it burdens the computational process. Topological queries can only be implemented at a very high computational cost (Frank *et al.*, 1989).

Analysis operations: Because of the hierarchical trees used in spatial occupancy enumeration reps, PMCs can easily be implemented. The PMC investigation is carried on individual cells and the result applies to the entire model. Boolean operators are well implemented and as result, new objects can be created. Surface areas and object volumes can be obtained from voxel dimensions. Cell decompositions cannot easily implement the PMC because of the potential of having multi-faceted cells, and this would mean that computations will be carried out between the line (in PMC) and a number of faces thus increasing the computational costs.

3.4 Sweep methods

The basic notion embodied in sweep representations is that of a point-set moving through space along a defined trajectory, and in the process, sweeping out a "volume", thereby, defining a 3-D object. Sweep-representations define 3-D objects according to a pre-defined set of rules (Samet, 1990, Streilein, 1996). Depending on the set of rules by which the object is created, the following classes of sweep representations can be distinguished.

3.4.1 Translational sweep:

The shape is translated along a pre-defined translational vector (Figure 3.13, below). In this class, the sweep's volume is the product of the length of the sweep and object's area.

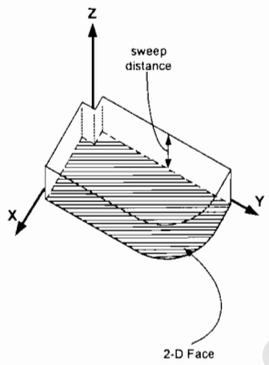


figure 3.13 Volume generated through Translational sweeping of a 2-D face

3.4.2 Rotational sweep

The shape is rotated around a pre-defined rotational axis. The swept object can be 2-D or 3-D. The Figure 3.14 below, shows a solid that has been generated through the rotational sweep of a 2-D object.

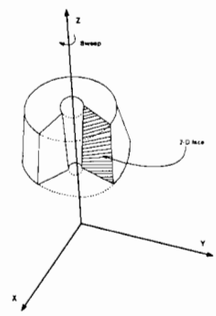


Figure 3.14. 3-D object generated from a rotational sweep of a 2-D face

3.4.3 General Sweeping

Very little is known about general sweep representations. Figure 3.15 shows that sweeping a curve may produce a set which is not homogeneously 3-D. (Note the dangling plane in Figure 3.14).If the sweep direction is parallel to the 2-D surface of interest, the resulting model will be 2-D as well. General sweeping may also involve the motion of non-rigid object, which undergoes deformations as it travels through space as is done in modelling systems based on the so called generalised cylinders (Agin *et al.*, 1976).

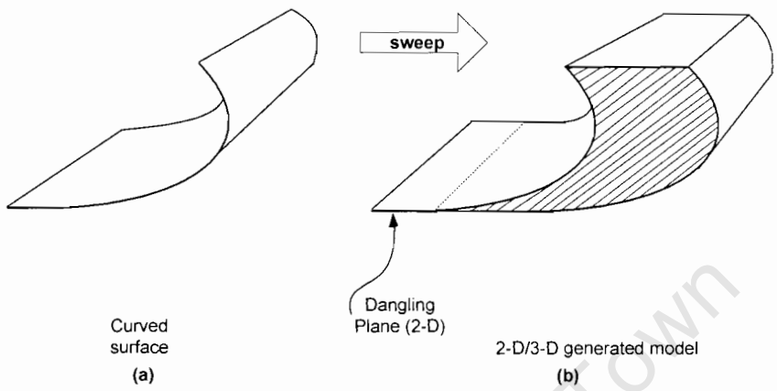


Figure 3.15: General sweeping may produce non regular sets

3.4.4 Minkowski-Sweeping

The Minkowiski sum of two sets ***A*** and ***B,*** denoted **A+B**, is the region swept by the set ***B*** when its reference point takes all positions within ***A***. Figure 3.16 illustrates this definition. The orientation of ***B*** is fixed during the sweep.

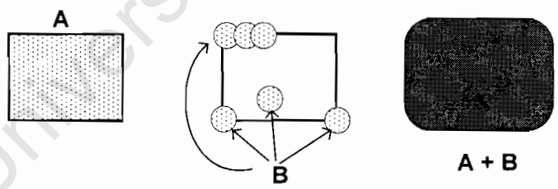


figure 3.16: Minkowski-Sweeping. On the right is a larger rectangle, A. The centre Figure shows the disk B at several positions it takes during the sweep. Solid object (right) is generated

If object ***B*** (see Figure 3.16, above) is replace by a sphere, a 3-D model is generated. When ***A*** is a 2-D planar set and ***B*** is a line segment normal to the plane of ***A***, their sum is simply a translational sweep or extrusion. Figure 3.17 shows an example. Extrusions are widely used in mechanical CAD systems (Rossignac *et al.*,1984).

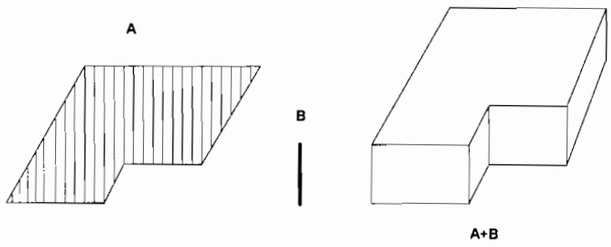


Figure 3.17. A Minkowski extrusion operation. The solid object (A+B) is Generated By B tracing the edges of A, in the process defining a volume.

3.4.5 Properties of sweep representations

The properties of transactional and rotational sweeping are determined largely by the objects class used to model 2-D or 3-D objects as well as the functions that describe the trajectories (Rossignac *et al.*,1984). Their general properties can be summarised as (Requicha, 1980):

Domain: Their domain is limited to objects with translational or rotational symmetry.
One can imagine the difficulty of building-up a complex object from sweep representations. Sweep representations are widely used in computer vision, since they provide an attractive way of constructing a limited range of objects for users of a geometric modelling system.

Unambiguity: Translational and rotational sweeping representations are unambiguous. This is a result of the fact that each model has a specific set of rules that defines the trajectory or sweep rotation path taken by the object during the formulation of the models. This ensures that no two objects will share the same model.

Uniqueness: The representation is not unique The sweep steps in model generation are not well defined as a result the different set of steps that could be followed resulting in varied models for the same object.

Validity: Sweep representations are prone to self-intersecting and as a result it becomes difficult to validate the models.

Conciseness

The algorithms for sweep operations are generally well defined and this makes the representations concise.

Ease of construction

The rotational and sweep representations are easily implementatable once the necessary algorithms are available. Many systems provide the user with the functionalities that allow the modelling of sweeps, although the models are often stored in a different representation format. However, the generation of arbitrary objects is difficult with this technique (Streilein, 1991).

Efficiency in the context of applications: In the manufacturing Industry, Algorithms to perform the following have been developed:

(1) Material removal: The effect of machining operation is to remove from a work piece the solid volume swept by the cutter when it moves along a specified trajectory (Voelcker *et al.* 1974)

(2) Dynamic interference: A solid ***S1*** moving through space collides with another solid ***S2*** if the volume swept by ***S1*** intersects ***S2*** (Boyse, 1979).

They have a number of limitations though:

- A major disadvantage of such a representation is the paucity of algorithms for computing the properties of represented solids.
- A lot of computational work is involved in the conversion to other representation formats during particular operations (e.g. storage)

The space requirements for the resulting 3-D objects are relatively lower, compared to B-reps and spatial partitions. In sweep reps, information relating to the original object and the trajectory function or sweep axis is stored. In most cases, the data relating to the original object is ***(n-1)*** dimensional, where the resulting model will be an ***n***-dimensional entity. In most sweep systems, the models are only generated into 3-D during visualisation, this ensures that space is conserved on storage as data remains stored as 2-D.

Topology: The sweep representations hardly store topological information. The generated 3-D models lack specific definitions of vertices, edges and faces, as a result capturing topological information is not possible.

Fundamental queries and operations: From the sweep trajectories, the volumes and surfaces areas of some of the resulting solids can be obtained. It is not easy though to perform metric analysis on the majority of the resulting solids. It is also difficult to apply regularized Boolean operations to sweeps without first converting to some other representation (Foley *et al.*, 1991).

3.5 Primitive instancing

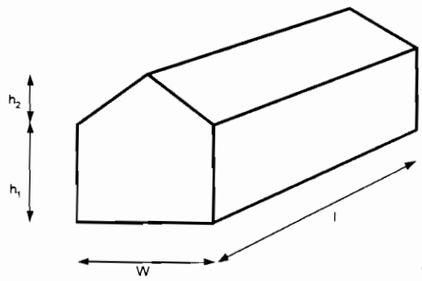


Figure 3.18: A simple primitive, this can be used for modelling buildings.

Primitive instancing provides a set of pre-defined object types. Each object type is represented by a small set of parameters, and individual instances are created by selecting a model type, providing the new instance with a parameter tuple and applying a rigid motion (a translation and a rotation in 3D space) to the primitive. Figure 3.18 shows a simple building primitive described by its length l, its width w, the gutter height h_1, and the roof height h_2. The primitive shape is described by the parameter tuple $\mathbf{P} = (l,w,h_1,h_2)^T$.

3.5.1 Properties of primitive instancing

Primitive instancing properties can be summarised as:

Domain: A predefined "library" of primitives provides the domain of the objects that can be modelled. Primitive instancing is often used for relatively complex objects, such as gears or bolts, which are tedious to define in terms of Boolean combinations of simpler objects (Foley

et al. 1991). Although a hierarchy of primitive instances could be built, each leaf-node instance is still a separately defined object. Primitive instancing is used in engineering work where complex structures are needed. In those instances, a primitive for the structure is developed.

Ambiguity: Primitive instancing models are unambiguous. Each object has a specific algorithm that generates the model, hence the non-existence of ambiguity. It should be realised though, that the non-ambiguity holds for a small domain of objects to be modelled, small enough to be covered by the family of objects in the database (Baumgart, 1975).

Uniqueness : Primitive instancing does not always guarantee uniqueness, for example, both spherical and elliptical primitives may represent a sphere. If however there is a thorough screening of primitives in the building of primitive-set, then the non-uniqueness could be avoided.

Validity: Conditions can be imposed to the parameters in order to achieve validity of all models, which can be created by the controlled variation of these parameters. For instance, none of the parameters of the primitive in Figure 3.17 can become zero or negative (i.e. $h_1 \geq 0$, $h_2 \geq 0$, $w \geq 0$ and $l \geq 0$). If for example, $h_2 = 0$, the primitive shape will change to a cube (or tetrahedral) and this will create an invalid shape according to the primitive set rules.

Conciseness: Primitive instancing representation is described as concise (Bosselmann, 1998). One basically needs a single primitive from the "family" database, parameters and an algorithm to describe the object model.

Ease of construction: Models can easily be generated through primitive instancing. However, the objects have to fall within the range of primitives represented in the database. The major weakness of primitive instancing is the lack of means for combining instances to create structures which represent new and more complex objects.

In primitive instancing, no provision is made for combining objects to form a new higher-level object, using for example, regularised Boolean set operations. Thus the only way to create a new kind of object is to write the code that defines it. The other drawback is the difficulty of writing algorithms for computing properties of represented

solids. A considerable amount of family-specific knowledge must be built into the algorithms, and therefore each object family must be treated as a special case, allowing no uniform overall treatment (Bosselmann, 1998).

Efficiency in the context of applications: Providing a large number of object types in a database of primitives requires an enormous programming effort, as tailored code is required for each object type. However, for the object types contained in the primitive database, modelling becomes easy and efficient. That is why primitive instancing is supported by many CAD and modelling systems as an auxiliary technique for the representation of often-needed parts, even though the modelling system itself is based on other representations. Primitive instancing is employed in systems that need high-quality graphics (Foley *et al.* 1991).

Topology: Because the representation does not consider different elements that constitute a body, topology capture is generally considered as poor (Brisson, 1989).

Fundamental queries and operations: Because of the parameters, which are readily available from the database, the following metric operations can be performed; surface area, volume and general edge dimension queries. The lack of topological information makes spatial analysis difficult (Brisson, 1989)

3.6 Functional representations

The most commonly used functional representations for modelling curved surfaces in computing are implicit and parametric patches. The parametric patches class is further subdivided into Bezier, Hermite and non-rational basis splines (Nurbs). Most manufactured objects, and many natural objects, are better modelled by combinations of functional representations (Watt *et al.* 1992).

3.6.1 Implicit Functions

Implicit surfaces are defined by a polynomial, P, of three variables

P (x, y, z) = 0, where x, y, and z represent co-ordinate values for the surface points.

The implicit representations are considered very difficult to use, although some CAD applications employ them (Frisken *et al.*, 1996). Detailed analysis of implicit functions will not be considered in this research, however reference is made to the literature which includes Farin G. 1997, Foley *et al.* (1983), etc.

3.6.2 Parametric representations

Parametric surfaces are defined by a set of three functions, one for each co-ordinate axis as follows:

$$f(u, v) = (x(u,v), y(u,v), z(u,v))$$

with parameters u and v in the range 0 and 1.

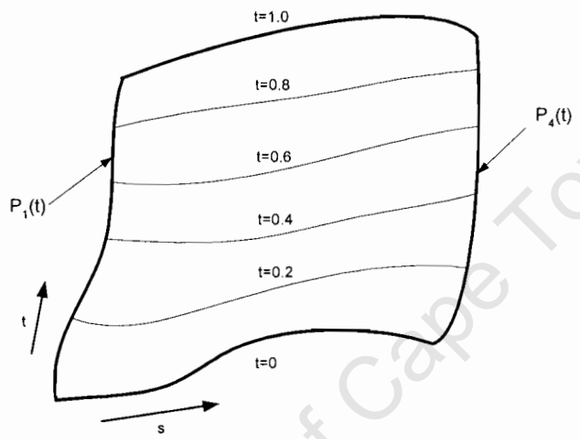


Figure 3.19. Lines of constant parameter value on bi-cubic surface: $P_1(t)$ is at s = 0, $P_4(t)$=1.

Parametric surfaces, or more precisely parametric surface patches, are not used individually (Farin G. 1997). Normally, many parametric surface patches are joined together edge-to-edge to form a continuous complicated shape. The parametric patch can be considered a union of an infinite number of curves. There are many ways of forming these curves and the so-called iso-parametric curve is amongst them. Given a parametric surface $P(s,t)$, if s is fixed and t allowed to vary, this generates a curve on the surface whose s coordinate is a constant (see Figure 3.19 above). Similarly, fixing t to a value and letting s vary, we obtain another iso-parametric curve(see Figure 3.19 above). Therefore, fixing t at 0, 0.2, …., 0.8 and 1, we shall have 6 iso-parametric curves. These curves sweep out the surface if we let t

range from 0 to 1 continuously. Similarly, the iso-parametric curves generated by varying **s** complete the surface. Figure 3.20 below, illustrates 3-D surfaces that are formed from iso-surfaces.

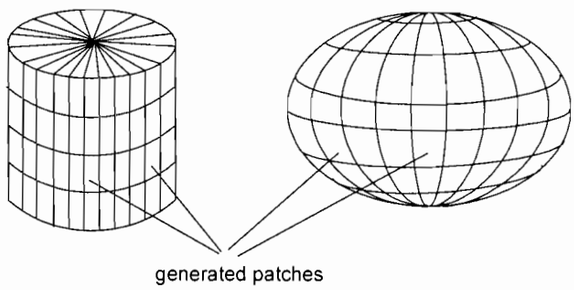


Figure 3.20: Using iso-parametric curves to generate 3-D surfaces

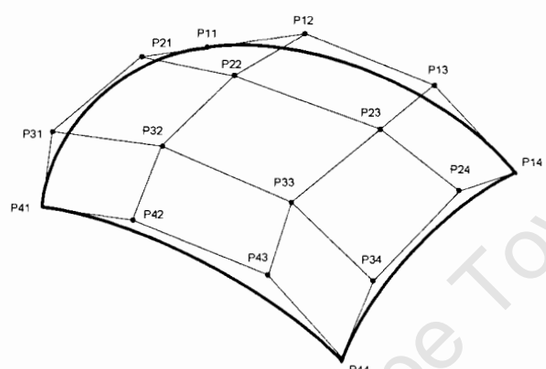


Figure 3.21. Control points for a bi-cubic, Bezier patch. The four corner points anchor the patch and will lie on its surface. The inner twelve points "pull" on the surface and control its shape in exactly the same way as the inner two points control 2D Bezier curves.

3D patches may use a variety of control strategies, including Bezier, Hermite, and B-Spline bases(Frisken *et al.*, 2000). Since the construction of different parametric surfaces is based on similar principles, only details of the Bezier surfaces are provided. The cubic Bezier curves require four control points with the end points controlling the curve position and the two inner points controlling the tangent vector at the end points. Bezier patches extend this control strategy in a logical fashion using sixteen control points. The four corner points control the position of the patch and lie on its surface (see Figure 3.21, above). The twelve intermediate points control the tangents to the patch along the edges and at each corner and may be used by the designer to "pull" the surface of the patch into the desired shape (Eck *et al.* 1996)

The mathematical definitions of other parametric representations are well covered in Farin (1997).

The Bezier form of the bi-cubic parametric patch has a very concise matrix formulation specifying a vector point, **P(s,t)**, on the surface in terms of the sixteen control points, $\mathbf{P}_{00}$... $\mathbf{P}_{33}$ as shown in the diagram above. This relationship is expressed as (for detailed analysis, see Farin, 1997):

$$P(s,t) = SBPB^{T}T^{T}$$

where parameter row vector $S = [s^3\ s^2\ s\ 1]$, parameter row vector $T = [t^3\ t^2\ t\ 1]$

T= [t^3, t^2, t, 1]

B is the Bezier coefficient matrix) and **T**= [t^3, t^2, t, 1], P is the control point matrix

3.6.3 Properties of functional Representations:

Domain: Parametric representations can model surfaces that are topologically equivalent to a sheet, a cylinder or torus (Eck *et al.* 1996). Nurbs allow for the modelling of conics and this extends the parametric surfaces domain. Operations involving implicit surfaces are regarded as complex and have a limited domain.

Unambiguity Parametric function models are unambiguous (Tokumasu *et al.* 1983). Each model is generated from a particular set of functions, matrices and algorithms, applied onto a particular object.

Uniqueness: Parametric functions are not unique .The functions for model are quite varied, as a result a number of different models for the same object can be generated.

Validity : Because surfaces are made up of patches, they have to be continuous. Continuity is tested and classified as C^0, C^1 and C^2 (and more generally C^n). The necessary conditions for C^0 are that the patches should join. For C^1 continuity, the control points along the edge and their tangent and twist vectors must have similar values (i.e. same positional co-ordinates, normals and vector matrices, respectively). And for C^2 continuity to exist there has to be equality for the normals along the edge of the two meeting patches. This rigorous testing implies that validity checking for parametric functions is a complex exercise.

Conciseness: The parametric representations are represented by well-defined concise functions (Chen *et al*, 1993).

Ease of construction: Tools for parametric modelling have been well established over time, and these allow the user easy interactive development of surfaces. Most of the tools developed have accompanying documentation, which enhances their use. Used mostly in modelling free-form surfaces, the Nurbs representation is attractive in interactive creation of objects, because (Chen *et al.*, 1993) it:

- represents infinite curves
- provides "local control" so that changing a control point affects only a local piece of the curve.
- can be made either to interpolate or to approximate control points, depending on application requirements.
- permits refinement by subdivision and knot addition, facilitating display and manipulation.

Efficiency in the context of applications: The iso-parametric curves are employed in some algorithms for rendering purposes. In other applications, the surface is triangulated (or polygonised or tessellated) into triangles or polygons. These triangles can then be rendered with existing graphics programming libraries such as OpenGL and PHIGS PLUS (Foley *et al.*1983).

Topology: Generally, functional reps are poor in capturing topological information (Foley *et al.* 1983). Parametric structures typically need associated data structures such as B-reps or spatial portioning structures to represent connectivity. Boolean operations cannot easily be used in analysis and new object creation.

Fundamental queries and operations: Parametric functions are efficient in metric operations (Chen *et al.*, 1993). The functions readily compute volumes, distances and areas.

Distance Fields in 3-D Modelling

A distance field is a scalar field that specifies a distance to a shape , where the distance may be signed to distinguish between the inside and outside of the shape (see equation, below).

$$D(p) = sign\ (p).\min\{|\ \ p - q\ |\}\quad q \in S$$

$$sign(p) = \begin{cases} -1 & if\ inside \\ +1 & if\ outside \end{cases}$$

Where S represents the object surface.

Distance fields are used for the object representation by several authors. Gibson(1998) shows how distance fields can be used to smooth out surfaces in segmented tomographic data by means of elastic surface nets. Breen *et al.*(1998 &1999) use distance fields to construct offset surfaces for superellipsoid models and to morph different geometric model types (polygonal meshes, CSG models and tomographic scans) in a single animation. A recent and powerful application of distance field representations is in volume sculpting, where significant steps in the creation of the so-called *digital clay* have been undertaken (Bremer *et al.* 1999).

The aim of distance modelling is to encode the object as a distance field. As illustrated in Figure 3.22, the minimum distance from grid point ***p*** to the object, point ***q*** in this case, is determined. In general, a 3-D grid is generated within object space and the grid line intersections provide the points ***p***. The computed distances are stored in a distance map (see Figure 3.22, below).

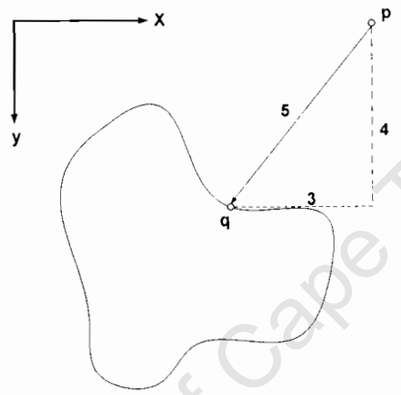
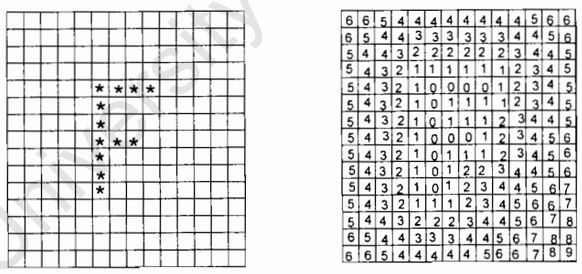


Figure 3.22 Distance Measurement: Two distance measurement approaches: Euclidean (arrow) and city block (dashed).



(a) **(b)**

Figure 3.23. Example of a Distance Map. To the left is the binary image, with feature (*) and non-feature (blank) pixels. To the right is a binary image: Each pixel has a value corresponding to distance to the nearest feature pixel. Note that the 0 (zero) values represent the feature pixels.

A distance map (Figure 3.23(b)) shows the storage of the distance field. Manipulation of the distance field becomes easier once a well-defined structure like the distance map is used for storage. The map ensures that the distances are readily available to the user or to algorithms in a structured manner. Examples of algorithms that are linked to the distance map include surface generation *marching cubes*, *central difference* operator and *the tri-linear interpolator* (see Chapter 5) (Schroder *et al.* ,1994).

The proposed distance map has some important properties (see Chapter 5):

1. The gradient of the distance map yields the direction of the surface normal and this is important to object rendering.
2. The zero-value iso-surface of the distance map is the object surface.
3. The following important operations can be carried out: locating surface points, performing inside/outside and proximity tests, Boolean operations, buffer operations, blending and determination of closest points on the surface. (see Chapter 5)
4. In addition to sampled distance values, grid point elements may contain other values like intensity, colour, opacity etc (Gibson, 1998).

3.7.1 Adaptive Distance Fields(ADFs)

Although the distance field is an effective means of representing shape, regularly sampled distance fields have drawbacks because of their size and limited resolution (Frisken *et al.* 2000). In order to capture fine object detail, an adaptive grid is employed. In other words, in areas of fine detail, the grid resolution is increased (see Figure 3.24a, below). In order to process the adaptive sampled data more efficiently, ADFs store the sampled data in a hierarchy for fast localisation. The combination of detailed directed sampling and the use of a spatial hierarchy for data storage allows ADFs to represent complex shapes to arbitrary precision while permitting efficient processing (Gibson 1998). Adaptively sampled fields were first described by Frisken *et al.* (2000).

Adaptive sampling permits arbitrary accuracy in the reconstructed field together with efficient memory usage.

In summary, ADFs consist of adaptively sampled distance values organised in a spatial data structure together with a method for reconstructing the underlying distance field from the sampled values.

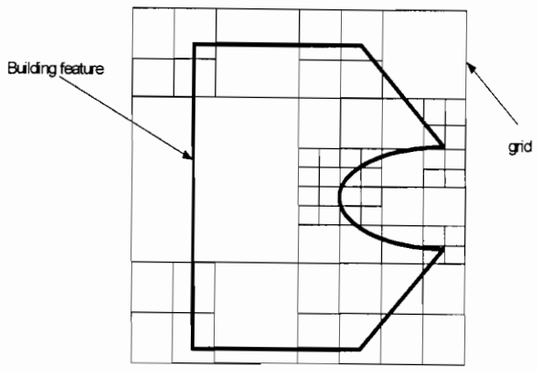


Figure 3.24a. Building feature in an Adaptive Distance Field. Note the increased resolution levels in areas of fine detail. (2-Dimensional case).

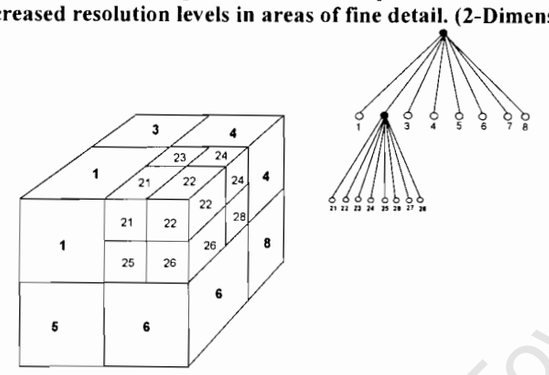


Figure 3.24b. Octree data structure for a 3-D grid

3.7.2 Octree based ADF

The ADF data is encoded in an Octree (see Figure 3.24b, above). Octree data structures are well known and in this section, we assume familiarity (see Chapter 5). Jones *et al.* 2001 shows that the use of octree data structures in distance field generation improves the distance calculations and data storage requirements.

In the proposed octree based ADF, each octree cell contains the sampled distance values of the cell's eight corners and pointers to parent and child cells. Given a shape as in Figure 3.24a above (2D illustration, hence a quadtree), subdivision of a cell depends on the variation of the distance field. The boundary cells are only subdivided when the distance field within the cell is not well approximated by the tri-linear interpolation of its corner values. Hence large cells will be used to represent edges where the shape is more regular and smooth, and this hopefully leads to data compression. Only sharp corners with high curvature provoke repeated subdivision.

3.7.3 Distance Metrics

There are several ways to define distances. Some of the distance metrics used in 3-D modelling are given below:

3.7.3.1 Euclidean Distance

This is popularly referred to as the "naïve" approach to distance computation. The distances are obtained from the Euclidean concepts;

$$D_E = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$$

The method for obtaining a true Euclidean distance field is to repetitively calculate the distance (see equation, above) between a grid point and every point on the surface, taking the minimum value as the result. This brute force method, is extremely computationally expensive (Satherly *et al.* n.d.).

3.7.3.2 Distance Transforms

Distance transforms (DT), also known as distance field calculations, devised by Rosenfield and Pfaltz (1966), approximate Euclidean distance via local distance propagation, thereby reducing the global operation to simple addition. This reduces the execution time considerably, but at the cost of reduced accuracy. The notable distance transforms are considered mentioned below (for a detailed analysis of the algorithms, see Appendix D):

City Block: Also known as the Manhattan distance. This metric assumes that in going from one grid cell centre to another, it is only possible to travel directly along grid lines (i.e. only vertical and horizontal steps are regarded). Diagonal moves are not permitted. Therefore the "city block" distance is given by:

$$dist_{CITY}(p,q) = |p_x - q_x| + |p_y - q_y| + |p_z - q_z|$$

where p and q are two points within the 3-D grid.

Chessboard distance: This metric assumes that "moves" can be taken both in diagonal and horizontal planes (i.e. considers vertical, horizontal and diagonal distances). The distance is obtained from:

$$dist_{CHESS}(p,q) = \max\{|p_x - q_x|, |p_y - q_y|, |p_z - q_z|\}$$

Since their original proposal DTs have been the subject of numerous research papers; see Danielson (1980), Borgefors (1984,1986 &1995), Ragnemalm(1993) and Mullikin(1992).

3.7.4 Properties of Distance Fields (DFs)

Distance field representation properties are summarised below:

Domain: The domain of Distance fields is quite large, ranging from detailed and complex object modelling to simple surfaces (Jones *et al.* 2001). Where object geometry (i.e. involving sharp edges) is required, DFs don't fare well.

Unambiguity: Ambiguity does not exist with DFs. Each model is created from a specific set of parameters that are "fed" into the modelling algorithm (Foley *et al.* 1991)

Uniqueness: There is some lack of uniqueness with DFs. The lack of uniqueness is mainly due to the fact that different grid resolutions can be used, resulting in a varied number of models for the same object (Jones *et al.* 2001).

Validity: Checking object validity is basically a question of investigating the field values. The values are empirical generated which implies that checking for non valid values will entail checking the distance field against the algorithms. All the models generated are a result of field values stored within the distance maps and generated from well-established algorithms, hence they are valid models.

Conciseness: The Distance fields representation structure is uncomplicated and but computationally involved (Gibson, 1998). A distance field of the object in question has to be generated using the appropriate algorithms and the resulting field, stored in a distance map. Using the matching-cubes algorithm (see Chapter 5), the surface of the object is generated. Hence, DFs are not concise.

Ease of construction: Algorithms for distance field generation (the distance transforms) and surface generation (e.g. matching cubes, see chapter 5) have been developed, so a 3-D model can be developed easily. The distance transforms have meant that the generation of the distance fields has been made easier compared to the "naïve" approach of using the Euclidean distances.

Efficiency in the context of applications: Frisken and Perry (2001) have used ADFs in creating virtual 3-D models. Distance fields have been applied to collision detection in robotics applications and representation of complexity. In Machining applications, the ability to model off-sets (i.e. buffer regions) from the distance field has enabled surface machining operations. Fractals and mathematically complicated shapes can be processed as quickly as much simpler shapes in Distance Fields graphic operations. ADFs have been successfully used in a Kizamu, a computer-based system for creating digital characters for the entertainment industry (see Frisken and Perry, 2001).

Topology: DFs are weak when it comes to providing topological information. Although the topological information could be stored within the grid points set of parameters, obtaining this information efficiently would be taxing. The process would involve instigating each grid point for the required topological data.

Fundamental queries and operations: Distance fields can be used for a number of important operations such as location surface points, performing the inside and outside and proximity tests, Boolean operations, blending and filleting, determining the closest points on to a surface and morphing between shapes (see Chapter 5).

3.8 Comparison of representations

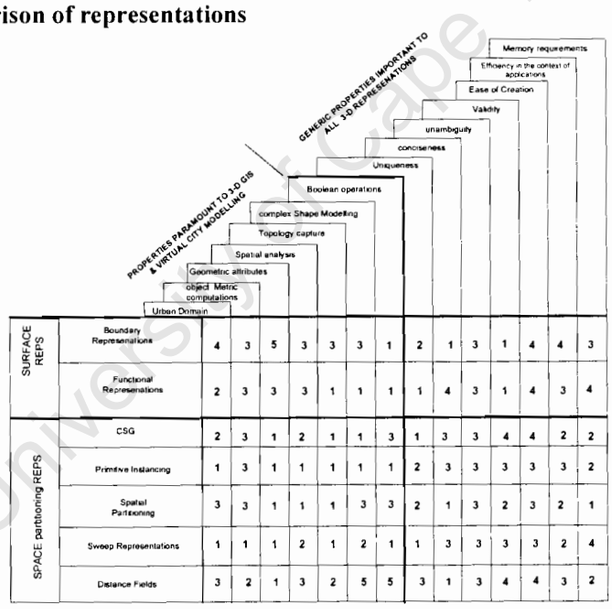


Table 3.2: A Summary of the properties of 3-D modelling representations. KEY 5=Excellent, 4=Very Good, 3=Good, 2=Fair and 1=Poor

Table 3.2 provides a summary of the representations' properties which have been reviewed in the preceding sections of this chapter. The quest is to come up with a representation that satisfies the stated urban requirements (see chapter 2). The analysis is grouped into two classes, surface reps (see section 3.81) and space partition reps(see section 3.82). In order to aid the analysis process, values are assigned to different qualitative descriptions of rep properties (Table 3.2, above).

3.8.1 Surface representations analysis

In the surface rep class, B-reps and functional reps are compared. An analysis of the properties which are paramount to 3-D urban modelling reveals that the B-reps (average value- 3.1) fare better than functional reps (average value- 2.0). Looking at the generic properties, both representations could be equally suitable (Note: Average value for B-reps is 2.6 , and average value for Functional reps is 2.8). Therefore, it can be concluded that B-reps are suitable for modelling the urban domain although they do have some weaknesses, as shown in Table 3.1.

3.82 Space partition representations analysis

For both set of properties (Table 3.1: properties paramount to 3-D urban modelling and generic properties), the DFs are the best modelling option with averages of 3.0 and 3.1. Compared to CSGs, the DFs have a better complex modelling ability and accommodate Boolean operations .

3.8.3 What is the representation for the research?

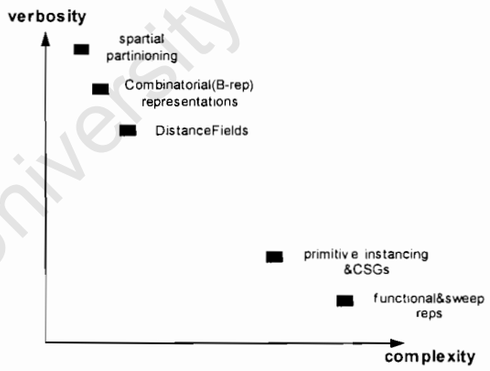


Figure 3.25..): Verbosity /Complexity trade off for 3-D representations. Modified from Bruce Naylor (n.d)

It is highly impossible to suggest a single representation that fully satisfies the modelling requirements of the urban environment. The analysis carried out points at having two representations, a B-rep and DFs.

Why should the two representations be ideal when they both have the "1" attribute value for the conciseness? Naylor (n.d.) states that, "computing with many pieces can be faster than attempting to process fewer but more complicated computationally complicated pieces". By alluding to Naylor's theory, it can be concluded that we have to trade a relatively small number of complicated operations for many more but simpler operations, i.e. verbosity/complexity trade-off (Figure 3.25).

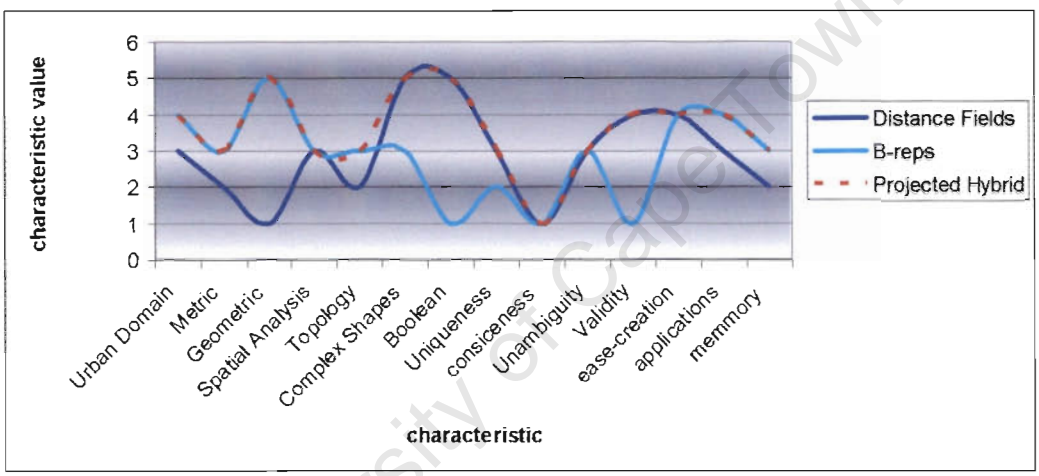


Figure 3.26. Plot of B-reps , Distance fields and projected Hybrid characteristics

Figure 3.26, above, provides analysis of the projected characteristics of the hybrid representation that will be the focus of this research. The resulting hybrid rep, at this stage, is basically a theoretical argument from the characteristics of the established representations. This research investigates in detail the design issues (see Chapter 5), and carries out a rigorous evaluation (see Chapter 7).

Chapter 4
The Distributed Internet Environment

This chapter gives an overview of the requirements for systems providing multi-user access using remote and/or slow networks (i.e. the Internet) and heterogeneous hardware (i.e. clients with different operating systems).

The main feature of a 3-D virtual urban city project is the need of a large amount of data on which to base the models (Hilary *et al.*, 1994). Typical to this problem of great quantities of heterogeneous data, is the need to allow immediate usability, easy transfer and movement. A range of innovative technologies is being developed that offers different ways of modelling and representing built-form and associated urban information with real-time interaction over the Internet. This research will present a review of the capabilities of the technologies and how they can be applied in the field of planning and design of urban environments. The World Wide Web (WWW) offers both an interface to, and a delivery channel for, the built environment information as well as being a medium for linking distributed users. We are interested in affordable "off-the-shelf" software, that is relatively easy to set-up and use and which requires standard PC- computing power preferable to a home user with a modem link (i.e., not high-end graphics workstations). The advantages and disadvantages, which these technologies offer, will be considered in terms of the level of realism and interactivity available to the end user.

4.1 The distributed system

What drives the need to distribute is summed up by Linthicum (1997) as the fundamental belief that "personal computers (PCs) connected to servers provide the best price/performance". In other words, we can do more with less, and thus bring computing power that was once out of reach to those who need it.

A distributed system is a program that consists of multiple parts running on more than one computer interconnected via a network (Bakman, 1995). This includes, distributed file systems, shared network printers, client/server, web-based systems, etc. There are many different models for distributed computing and the terminology and components vary between the models. Firstly, the "parts" of a distributed system (or "distributed application") may be

arranged and may cooperate in different ways. One common structure is organization as a "server" and a number of "clients". Hence, the "Client/Server" model (Linthicum, 1997).

4.2 Client/Server

The Client/Server, at its core, provides the ultimate "freedom" of mix and match software components at the client, server and all points in between (Edwards *et al.*, 1997). Rather than integrate applications and data on large centralized processors, we now take a "divide-and-conquer" approach to meet our information technology requirements.

Client/Server allows developers to split the processing load between two logical processes: the client and the server. Although both the client and the server may exist physically on the same computer, most of the client/server systems run the client process on one computer and the server on another. A network connection between the client and the server allows the server to exchange information as required by the application (see Figure 4.1, below).

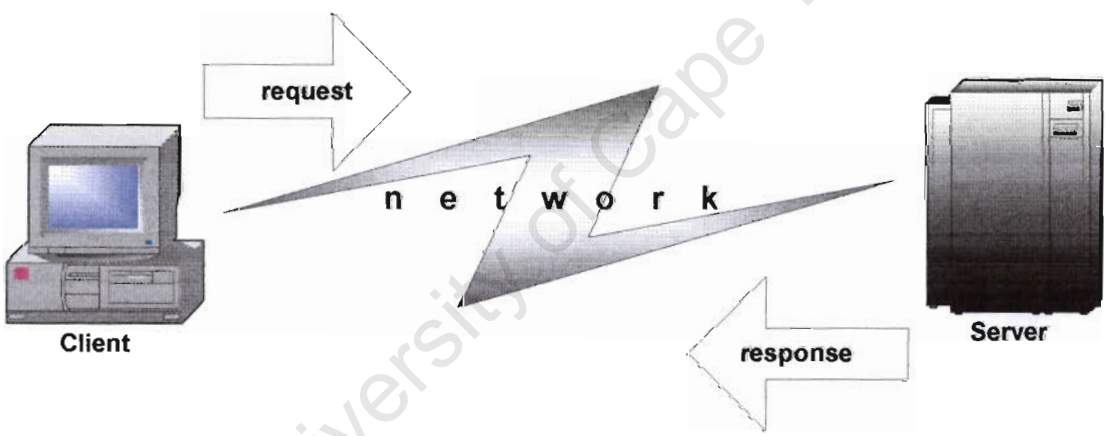


Figure 4.1: basic client/server architecture

In the client/server paradigm, servers come in four basic forms:

File servers: File servers provide files to clients as needed. Clients, access files on file servers as if they were stored on local disk. Examples of file servers' software include Novell's Netware and others.

Application servers: These are specialised computers that exist on the network and perform application-processing services. Typically these computers run transaction-processing (TP) monitors, distributed objects, or custom built processing services. Application servers allow developers to place common application services in a central location and remove the

application load from the client. Application servers also provide a mechanism to support the three-tier client/server architecture (see 4.2.1.2, below).

Database servers: Data servers are computers that exist on a network and provide data processing services upon request from the client. For instance, a client may send a database request in form of Structured Query Language (SQL). The database server processes the database request on behalf of the client, sending only the requested information back to the front-end application. Database servers run software such as Oracle, Informix, Microsoft windows2003, MySQL and Sybase.

Web servers: Web servers are computers that exist on the Internet or intranets that provide HTML documents, or graphics, video, or even database services upon request from a client running a web-browser.

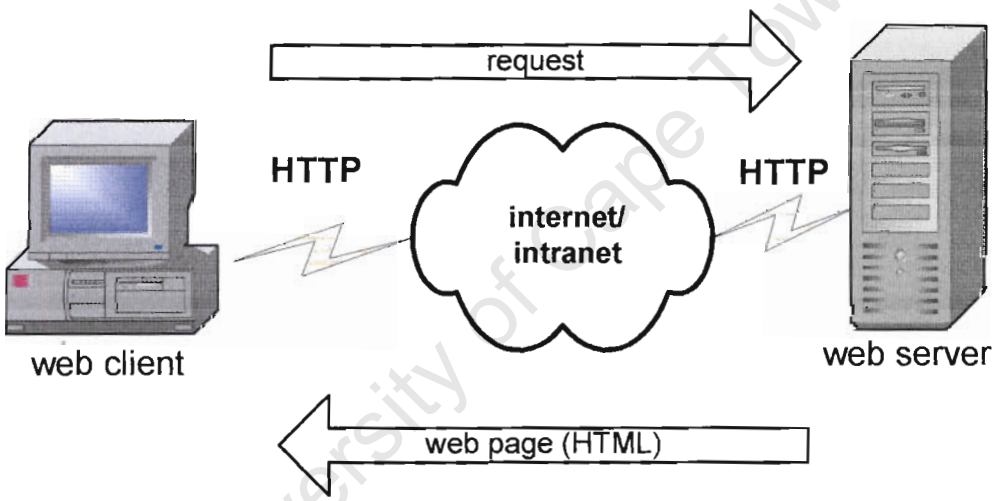


Figure 4.2. Web server architecture

The web browser (see Figure 4.2, above), links to the web server using TCP/IP network protocol and HTTP request content (HTML document, graphics, video, etc). The web server sends the content to the browser that presents it to the user. Webs server software examples include Netscape Enterprise Server and Microsoft's Internet Information Server.

4.2.1 "Fat" vs. "Thin" Clients

Historically, there were two "schools of thought" concerning the division of work between client and server (Martin et al., 1995). One advocated the use of "Fat Clients" The idea was to put significant processing in the hands of the client and thereby offload work from the server (which was often overloaded). Unfortunately, this meant that greater capability was required

of the clients machines .The use of "fat" clients also means there is more functionality at the client to configure and maintain and this translates to higher system administration costs. Another school promoted "Thin Clients". The idea is to limit clients to "display" functions e.g. a web browser as the client This addresses cost and complexity at the client but relies on a large and expensive server system that may not scale as the system grows. It also limits overall system functionality and there is a cost to go "to the server" on each action. So which one should we choose? There are at least three factors to consider when deciding how to distribute application functionality, at the client or server (Boar, 1993):

- Location of resources being accessed
- Communication costs
- Workload distribution

Given the choice of dividing up processing in one of two ways you should choose the one which minimizes communication This is because the speed of communication over the network is orders of magnitude slower than the speed of computation (Boar, 1993). When there is a "resource" that is available at a single machine, then the processing that uses that resource should be done where it is e.g. It makes sense to do database processing on the machine that hosts the database system.

Applications which are used by big organisations generally consist of three layers of logic (Chorafas, 1989):The "data access" layer, The "application or business logic" layer, and The "presentation" layer. The data access layer is responsible for managing and providing access to the long term ("persistent") data used by the application i.e. the data access layer handles the database(s). The presentation layer is responsible for presenting results to the users in a form that is easily understood by them i.e. the presentation layer handles the GUI of the application (e.g. forms, reports, etc.). The business logic layer does everything else i.e. the business layer actually implements the application logic to accomplish the necessary business function. Depending on how an application's layers are divided up among different machines, we can end up with a computing system which is: *two tier, three tier or N-tier.*

4.2.1.1 Two-Tier architecture

The first generation of client-server architectures is called the two-tiered (Gray *et al.*,1993). It contains two active components: the client, which requests data, and the server, which

delivers data and application services (see Figure 4.3, below). Two-tier applications tend to push a lot of processing onto the client machines. This architecture poses several problems:

- Client-side resource requirements balloon with the extra processing needs. It is not uncommon to find business applications requiring Pentiums with high megabytes of random access memory (RAM).
- User interface and business processing tend to get rolled together, especially with the rapid application development tools on the market. With the user interface so closely tied to business processing, changes on one, end up having a direct impact on the other, making maintenance difficult.
- With all this redundant processing occurring on many client machines rather than in a central location, new applications are forced to reinvent the wheel when dealing with the same business processing.

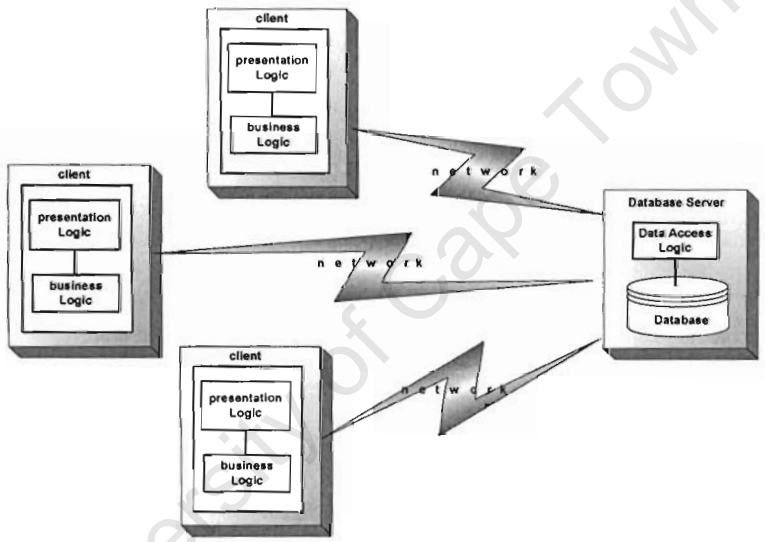


Figure 4.3. Two-tier architecture. Note that in this instance, the business logic is resident on the client side.

4.2.1.2 Three-tier architecture

A three-tier application is one where a third application layer exists between the client and server layers of traditional two-tier client/server development (Edwards *et al.* 1997). This middle layer has a wide variety of uses depending on the application in question.

The three-tier architecture uses the middle layer to separate business processing from the visual representation of data. This layer, called the application server, is responsible for knowing how to find and manipulate business data. The client evolves into a much leaner

application responsible only for retrieving information from the application server and displaying it on the screen.

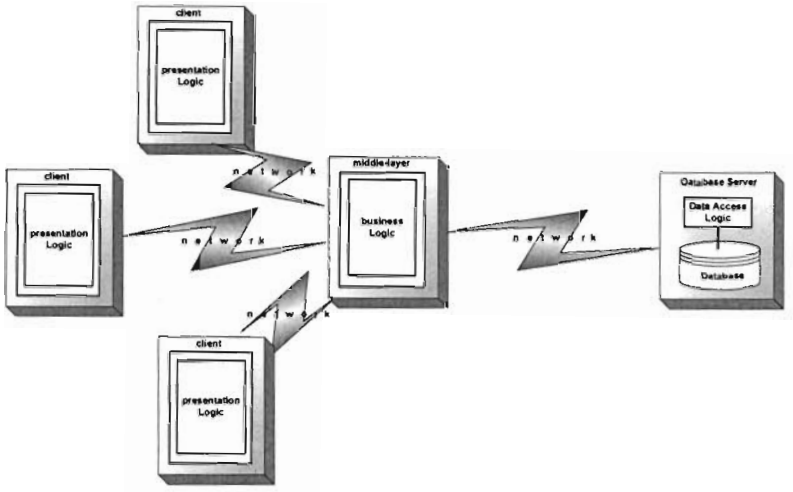


Figure 4.4. Three-tie architecture showing that the business logic is implemented in the "middle layer" server.

In addition to removing a huge processing burden from client machines, this application server can be used to consolidate enterprise-wide business rules. Where business rules had to be rewritten for each two-tier application thrust upon the desktop, application servers process business rules in a single place for use by multiple applications. When the business rules change, a change to the application server takes care of that change for all the applications being run by the business.

Of specific interest to many developers is the ability to hide any knowledge of the database server from the client. Additionally, this back-end independence enables applications to scale much easier across CPUs. Figure 4.4 above, shows a three-tier architecture.

Another advantage to using three-tier architectures for your applications is processing speeds. In other client-server applications, the application becomes a fat-client meaning the amount of CPU processing on the client becomes excessive. Using a three-tiered system allows most of the data processing to take place on the application server side, hence a thin-client application.

4.2.1.3 N-Tier architecture

Not every application will divide conveniently into presentation, business, and data access processing. Instead of the middle-tier being a monolithic program, it can be implemented as a

collection of components that are used as variety of client-initiated business transactions (Gray et al.,1993), see Figure 4.5 below.

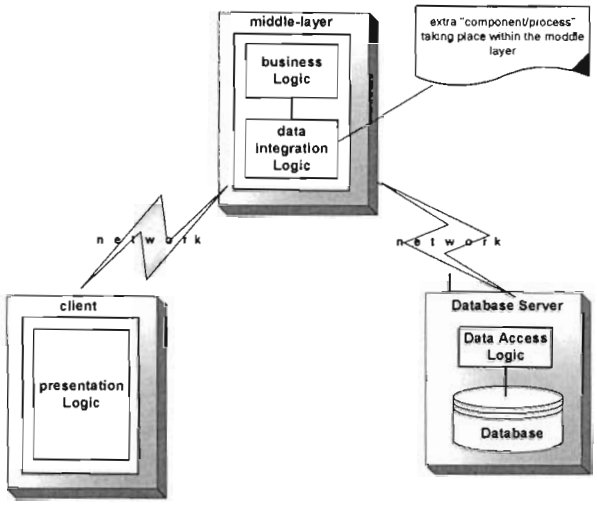


Figure 4.5. N-tier architecture showing "extra" components that can be added to the middle Layer server..

Each component automates a relatively small business function. Clients then combine several middle-tier components within a single business transaction. A component has the capability of calling other components to help it implement a request. Generally, it can be concluded that most of the *3*-tiers are really *N*-tiers.

4.3 User interaction within the distributed environment

Static HTML is fine for displaying relatively static content; but the challenge has been to create interactive web-based applications, in which the content of the page is based on a user request or system status, not pre-defined text (Burdea *et al.*,1993).

An early solution to this problem was the use of the common gateway interface (CGI) programs (Martin et al., 1995).

4.3.1 Common Gateway Interface (CGI)

The traditional way of adding functionality to a Web Server is the Common Gateway Interface (CGI), a language-independent interface that allows a server to start an external process which gets information about a request through environment variables, the command line and its standard input stream and writes response data to its standard output stream (see Figure 4.6, below). Each request is answered in a separate process by a separate instance of

the CGI program, or CGI script (as it is often called because CGI programs are usually written in interpreted languages like Perl) (Edwards,1997).

CGI programs are most often written in Perl or C++. The vast majority of web servers can not run a CGI program concurrently which means that each person who is waiting for that script must wait until the previous execution has completed. Each time someone starts a CGI script, a new process is created on the server; and if the script is written in an interpreted language like Perl, the server has to start up another Perl interpreter, taking up processing time and memory. Hence, traditional CGI programs have inherent performance and scalability problems.

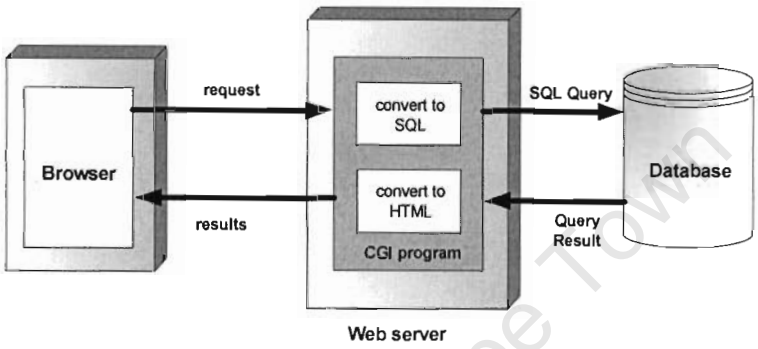


Figure 4.6. CGI architecture. Browser initiates request, Web server receives the request, For each request, web server spawns a new operating system process to execute the CGI Program.

4.3.2 Active Server Pages (ASPs)

Microsoft attempted to address the problems faced by CGIs when they introduced Active Server Pages (ASPs) (Kauffman *et al.*, 1999). ASPs are server-generated pages, which can call other programs to do things like access databases, serve different pages to different browsers - basically, anything that was done with CGIs. The active server pages enjoy most of the benefits that are displayed by the *Servlets* (see section 4.3.3, below) but have the biggest limitation in that they are confined to Microsoft's Internet Information Server (IIS), making them platform dependent.

4.3.3 Servlets

Servlets are the Java equivalent of other technologies such as CGI or ASP (Hall, 2003). The web server does all of the executions and calculations. The client browser is not doing any more work than it would to process a normal HTML page. In fact,

what the web server returns to the browser is virtually indistinguishable from normal HTML except that it has been generated dynamically.

Java Servlets have overcome limitations faced by CGIs. The following are their main advantages over CGIs (Hall, 2003):

Efficiency: For each browser request, the servlet spawns a light weight thread[1]. This is faster and more efficient than spawning a new operating system process. Hence, servlets have better performance and better scalability than traditional CGI.
Convenience: Servlets include built-in functionality for; reading HTML form data, handling cookies, tracking user sessions and Setting HTTP headers

Powerful: Servlets can talk directly to the web servers and multiple servlets can share data.

Portability: One of the advantages of Java is its portability across different operating systems. Servlets have the same advantage .You can therefore write your servlets on windows operating system, and then deploy them on UNIX. You can also run any of your servlets on any Java-enabled web server, with no code changes.

Secure: Traditional CGI programs have a number of known security vulnerabilities.
Hence, you usually need to include a separate Perl/CGI module to supply the necessary security protection. Java has a number of built-in security layers. Hence, servlets are considered more secure than traditional CGI programs.
Inexpensive: You can download free servlet kits for development use.

4.3.4 Java Server Pages (JSPs)

JSP is a relatively new extension to the Servlet Application Programming Interface (API) that makes writing servlets even quicker and easier. Servlets have two related drawbacks that JSPs address (Hall, 2003). First, it can be quite difficult to "see" and edit the HTML that is being sent to the client since it is embedded within Java statements. Second, Servlets were not

[1] A thread is similar to a sequential program i.e. it has a beginning, an end. However, a thread itself is not a program--it cannot run on its own--but runs within a program. Java has the ability to run multiple threads at the same time.

something that the average web master could easily maintain after they were created. JSPs use tags within an HTML page to perform the dynamic processing. Each time the web server sees a JSP it automatically compiles it into a servlet and executes it as such. Thus, JSPs make it much easier for the programmer to work with the HTML elements of the page and also makes it easier for others to modify the HTML elements.

4.4 Database interactions

One of the most challenging issues in distributed systems is to "talk" to databases that are hosted on remote machines (Burdea *et al.*,1993).

4.4.1 Open Database Connectivity

ODBC, a C-based interface to SQL-based database engines, provides an interface for communicating with a database over the networks. ODBC was created by Microsoft for Windows databases, but this has since spread to databases on most platforms (Whiting et al., 1996).

4.4.2 Java Database Connectivity (JDBC)

JDBC is a Java API that enables Java programs to execute SQL statements (Van Haecke, 1997). The interface allows Java programs to interact with any SQL-compliant database. Since most relational database management systems (DBMSs) support SQL syntax, and Java can run on most platforms, JDBC provides the flexibility to write a database application that can run on different platforms and interact with different DBMSs. Therefore, JDBC is ideal for internet/intranet application design.

Another standard part of Java is the JDBC-to-ODBC bridge, which is a layer of code that implements the JDBC driver API by calls to the ODBC driver API. You can use this bridge and an ODBC driver, instead of a JDBC driver, to provide access to any database with an ODBC driver.

4.4.3 JDBC versus ODBC

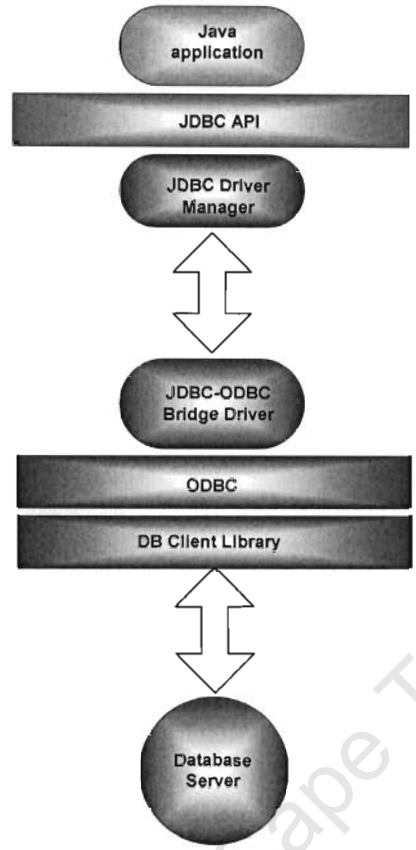


Figure 4.7. Showing ODBC/JDBC "bridge" connection. The Java application is sending request for data.

Microsoft's ODBC API is probably the most widely used programming interface for accessing relational databases. It offers the ability to connect to almost all databases on almost all platforms. So why not just use ODBC from the Java environment? ODBC can be applied within the Java environment only with JDBC to form the JDBC-ODBC Bridge (see Figure 4.7 , above). The following are the reasons as to why we cannot have ODBC to operate entirely on it's own in a Java environment (Van Hacke,1997):

- ODBC uses a C interface. Calls from Java to native C code have a number of drawbacks in the security, implementation, robustness, and automatic portability of applications.
- A literal translation of the ODBC C API into a Java API is not possible because of the differences in programming syntax.
- When ODBC is used, the ODBC driver manager and drivers must be manually installed on every client machine, while JDBC code is automatically installable, portable, and secure on all Java platforms from network computers to mainframes.
- The JDBC API can easily support both two-tier and three-tier models for database access.

4.5 Interacting with the 3-D Model

On the client-side (the browser), the quest is to interact with the 3-D model. Interactive operations include change of attribute values, making neighbourhood selections and the results showing graphically, generating visibility lines as well as on-line dynamic 3-D model generation.

For communication between a Virtual Reality Modelling Language (VRML) generated "world" and the external (user) environment, an interface between the two is needed. This interface is called an External Authoring Interface (EAI) (Leclerc, 1997).

4.5.1 The External Authoring Interface

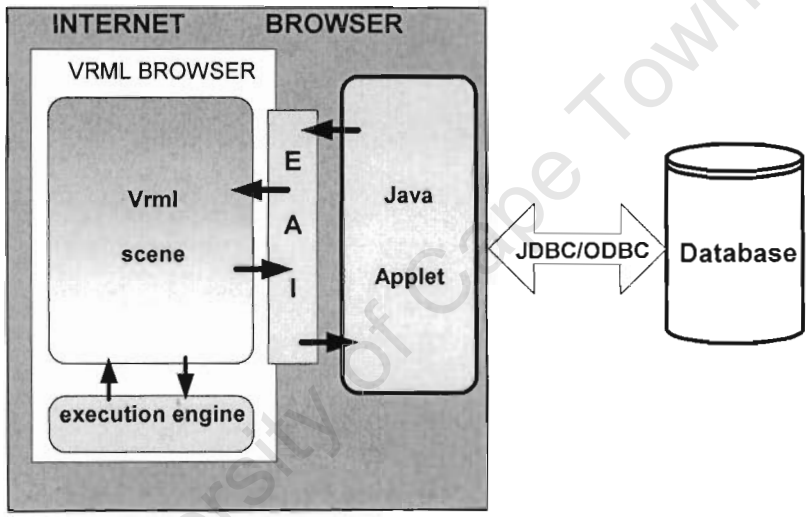


Figure 4.8. Overview of the Communication within the network browser

The EAI defines the set of functions on the VRML browser that the external environment can perform to affect the VRML world. The only practical example of communication between an external environment and a VRML world shown to date is the interface between a Java applet on an HTML page and an embedded VRML world on that same page (see Figure 4.8, above).

Chapter 5
The Hybrid 3-Dimensional representation (H3-D rep)

Chapter 3 compares the existing data structures in 3-D modelling and provides a summary of their properties. The analysis is grouped into two classes: solid and surface representation.

This review of the existing data structures leads to the main research question of this work; what is the ideal data structure for a complete 3-D representation in city modelling applications? The analysis suggests a hybrid of boundary representation (B-rep) and distance field modelling. This chapter provides the design of such a hybrid representation.

Spatial Operation	Brief description	Implementation H3-D component
Static Topological query	Topological queries enable the determination of spatial relationships between primitives making up a body and between feature objects and models.	B-rep
Interactive location based queries	Querying the distance from a point to nearest location for reconnaissance purposes in planning operations	Distance field
Static Metric Operations	Querying object dimensions	B-rep
Interactive Buffer Operations	Region generation for the purposes of analysis around a spatial feature	Distance field
Interactive PMC	For a point in space, the user can classify whether it is on the surface, inside or outside a particular body	Distance field
Interactive LMC	Like PMC, the user wants to investigate whether a line is inside, outside or on an objects boundary	Distance field
Visibility	Visibility analysis is important in establishing a clear line of sight.	B-rep

Table 5.1. 3-D spatial operations

The proposed hybrid representation is a fusion of distance field and boundary representation components. 3-D visualisation and spatial analysis operations are shared between the two components, exploiting the best features of both. The B-rep component implements the

general "static" spatial queries, whereas the distance field is ideal for "interactive" querying. General 3-D visualisation is implemented within the B-rep component.

Static queries are those queries carried out when the user make queries based on only the data stored in the system databases. Examples of these queries include are provided in Table 5.1 above. On the other hand, Interactive queries allow the user to define new spatial features and create queries between the feature and the existing features stored in the database.

This Chapter is structured as follows: Section 5.1 gives an in-depth review of existing B-reps. Section 5.2 describes the proposed hybrid 3-Dimensional representation and provides a detailed breakdown of the contributing components to the H3-D representation. The H3-D formalism and E-R mappings are then discussed in sections 5.3 and 5.4. Section 5.5 covers the spatial analysis operations that can be implemented through the Distance Fields component.

5.1 Design Considerations

In the conceptual design of the hybrid representation proposed in section 5.2, the visualisation and static query functions will be carried out by the B-rep component and dynamic queries by the distance field component. In this section, design considerations for the B-rep component, based on the weaknesses of previous B-rep structures are analysed. The analysis will be based on the following issues:

1. Space complexity: The representation should be able to conserve space as this is crucial in the face of massive data sets expected for geographic 3-D models

2. Time complexity: Querying being of fundamental importance, the representation has to be efficient as far as static queries are concerned. Comparisons made in this regard remain unclear among competing 3-D representation designs. Time complexity determination has adopted a "stop watch" approach, where programmes running on different machines are monitored. The "stop watch" approach entails monitoring all the hardware resources which include cache memory, random access memory, taking note of the central processing unit among many variables and this is no easy task. Time complexity comparisons will be based on queries ***T1-T10*** (see point 5, below).

3. Object representation: The representation must capture both manifold and non-manifold urban features. The domain of these features must include points, lines, surfaces and volumes (or bodies).

4. Hybrid suitability: The representation must link to the distance field component without requiring significant data transformation. This is very important as the thrust of this research is towards the creation of a hybrid data structure. As such, analysis will investigate these representations that pass on "complete" feature descriptions rather than deriving them from the information maintained. The representation must provide explicit feature definitions.

5. Spatial analysis: The representation will be tested against a number of topological based queries (***T1-T10***), listed below: (note: arc and edge will be used interchangeably). The major reason for analysing the specific queries ***T1-T10*** is that they are basic queries in spatial analysis. Major and complicated queries can be built by coming up with different combinations from these basic queries.

T1: Given vertex V_i, find all the edges linked to it.

T2: Given an arc A_i, find all faces incident on it.

T3: Given a face F_i, find its bounding arcs.

T4:Given a face F_i, find the vertices bounding the face.

T5: Given a face F_i,find bodies sharing it.

T6: Given a body B_i, find all the bounding faces.

T7: Given a body B_i, find all the bounding vertices

T8: Given a body B_i, find all the bounding arcs.

T9:Given a line L_i, find all the constituent arcs.

T10: Given a line L_i, find all the constituent vertices.

The following parameters will be adhered to:

Number of edges = N

Number of faces = M

Number of vertices = v

Number of bodies = b

Constant reading time = K

Based on the representation, algorithms are derived for each of the queries and the algorithms become the basis of time complexity analysis.

5.1.1 B-reps under review

A review of well-known B-reps is given below:

Basic winged edge data structure: The winged-edge data structure has been described extensively in literature and in Chapter 3 of this thesis. This representation has been the reference to the development of many B-reps (Woo, 1985).

1. Space complexity:

The winged edge data structure stores, for each oriented edge, eight references: two vertices, two faces, and four incident edges that share the same faces and vertices (see Chapter 3). In evaluation (chapter 7), it was discovered that the space complexity for the winged edge is Order $O(N)$, where N is the number of geometric primitives (i.e. the problem size). The storage requirements for the winged-edge and its variants are generally regarded as similar (Kettner, 1998).

2. Time complexity (efficiency):

In deriving a topological description, an edge is traversed twice (i.e., in opposite directions). With this data structure, one can easily answer questions: which vertices, edges, faces are adjacent to each face, edge or vertex. These queries can be done in constant time. The winged edge is considered the most efficient in traversal operations compared to its variants (Order $O(N)$, see chapter 7) (Kettner,1998). Chapter 7 evaluates the efficiency of the winged-edge for a specific set of query operations and was the most efficient among the competing B-reps for ***T2*** (i.e., constant time $O(2K)$) and ***T3*** (i.e. constant time $O(5K)$) queries (see Table 7.7, Chapter 7). This is expected as the strength of the winged-edge is on vertex/edge and edge/face topological relationships.

3. Object representation:

The fundamental primitive for feature modelling is the edge. The condition of each edge belonging to exactly two faces limits this representation to manifold objects. The representation of holes is achieved by changing the ordering of edges, i.e. anticlockwise for holes. The relational logical model for the winged-edge has three tables with the following

entity types: edge, vertex and faces. This would mean that bodies and lines are implicitly defined, making it a challenge to derive them during spatial queries.

The two symmetric parts in the winged edge correspond to two possible orientations of the edge (Kettner,1998). The inefficient case distinction in the traversal computation results from the fact that a pointer to an edge does not encode the current orientation context. Splitting the edge into two symmetric records, called half-edges and adding mutual links to each other, can solve the orientation problem. In both situations, the half-edge contains a pointer to an incident vertex, an incident face and the opposite half-edge. The structure additionally stores a pointer to the next clockwise half-edge and a pointer to the previous counter-clockwise half edge around the face. The use of the extra edge record, makes it less efficient. The inability to model non-manifolds is solved by the introduction of a variant, the radial edge (see Chapter 3), which allows more than two faces to be referenced by an edge.

4. Hybrid suitability:

Because bodies and lines are implicitly defined, it basically means that the features will not be readily available for passing on to the distance field during the mapping operation.

5. Spatial Analysis:

The winged-edge representation could easily implement the ***T1-T4*** topological queries (see Chapter 7). It is difficult to directly implement the other queries i.e. ***T5-T10***. The queries tested against are only a sample of the vast number of possible queries. However, they are a basis from which complicated queries are built. The number of queries that can be implemented translates to 40% of the sample.

The Formal Data structure (FDS): The Formal data structure was developed by Molenaar (1990) and its theoretical foundations are based on the winged-edge rep.

1. Space complexity:

The space complexity of 3D FDS is evaluated as $\boldsymbol{O}(N^2)$ (see Chapter 7). This places the 3D FDS as one of the few representations with high space costs, an inference arrived at by Zlatanova (2000) as well. This is expected because of the presence of the edge primitive, the node-in-body singularity, shape information and the "left" and "right" attribute values for the face (see Figure 5.1, below). The logical entity-relationship modelling gives rise to more data tables as well. The representation captures a high volume of attributes in its entity tables (see Molenaar, 1992).

2. Time complexity :

Lack of the explicit relationship, "face-part of a body", impacts on the computational time (*T6* Query). According to the representation, a query to find faces making up a body would mean traversing through the tables in the fields for "body left" and for "body right" attribute values. The edge primitive leads to a further interrogation of tables in all topological queries, which are edge-based (e.g. line, face, surface and body).In the evaluation carried out in Chapter 7, 3D FDS performs badly in most of the queries. The worst performance case is Query *T7* (i.e. Order O($54\boldsymbol{N} + 2\boldsymbol{Z} + 30\boldsymbol{K}$)) and this can be attributed to that fact that the query involves a number of entity types (i.e. body, edge, arc and node)

3. Object representation:

The 3D FDS (Figure 5.1, below), is developed along the same lines as the general B-rep structure as a direct consequence of the B-rep conceptual view (see Figure3.1, chapter 3). The 3D FDS consists of three main levels: class (related to a thematic class), four feature objects (point, line, body and surface) and four primitives (node, arc, face and edge). The formalism of the structure is governed by 12 conventions (see Molenaar, 1992). Importantly, the conventions project the structure as single valued, i.e. space is partitioned into non-overlapping objects. A primitive is can only appear once in the description of a feature object of the same dimension. This ensures a 1:1 relationship between a primitive and feature of the same dimension. Primitives of different dimensions however can overlap: for instance, node-on-face, arc-on-face, node-in-body and arc-in-body relationships are stored explicitly.

The representation of holes is very important. The direction of arcs will give an indication as to whether the relationship is that of face containment or a hole contained within a face. The hole will have an opposite orientation to the face.

4. Hybrid suitability:

The representation has at its disposal the surface, body, line and point features. These can be easily "read" into the distances field component of the proposed H3-D. But the representation lacks topographical information (DEM) and texture.

5. Spatial Analysis

With the 3D FDS, and based on the algorithms that are derived for the structure (see Appendix B) all the sample queries , ***T1-T10***, could be implemented.

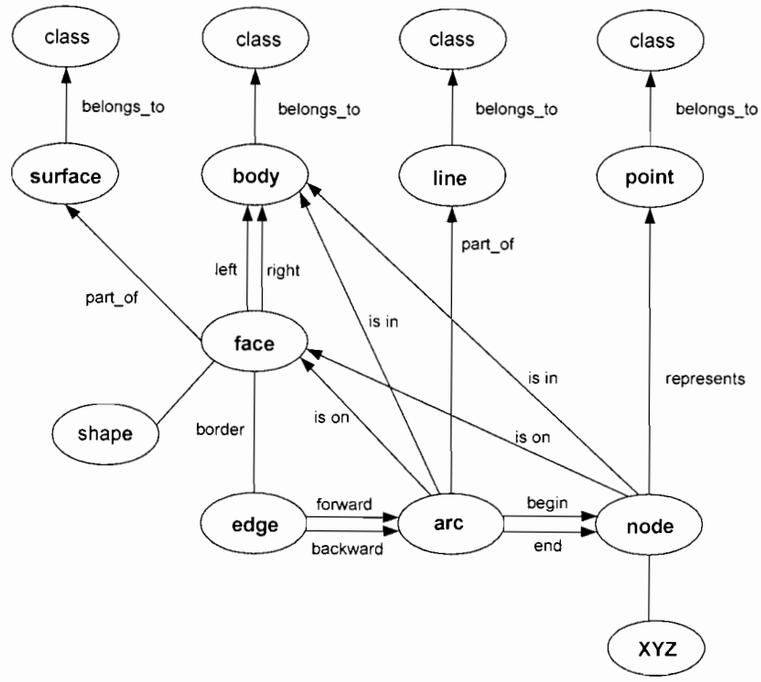


Figure 5.1. F3-D Formal Data Structure (3DFDS) (adapted from Molenaar,1998)

Urban Data Model Representation: Like the Molenaar's 3D FDS, the urban model (Coors,2003) is built from the winged-edge rep theoretical foundations.

1. Space complexity

The space complexity for the UDM (Figure 5.2)representation is $O(N)$ (see Chapter 7). Although the UDM lacks the arc primitive, the representation has a geometry feature and the logical design (Entity-relationship modelling normalisation, see Howe, 1989) produces more record tables. The partition of the objects gives rise to high space for data storage negating the savings made by not storing the arc primitive. All surfaces have to be triangulated.

2. Time complexity(efficiency)

In the limited number of queries possible, the UDM performs relatively better for ***T4***, ***T5*** and ***T10***. This is partly due to the fact that there is a direct link between the vertex and that the face feature tables and the face table contains the attributes Left-body and right-body, making query ***T5*** highly efficient.

3. Object representation

The representation is built on a full partition of the space and represents the geometry of a body or a surface by planar convex faces (Coors, 2003). Each face is defined by a set of

nodes. There is no direct support of an arc primitive. Two convex planar surfaces are adjacent if they share the same two nodes. In the relational representation, every face having more than three nodes is decomposed into triangles. *Singularities* are reduced to *node-on-face* and *arc-on-face*. The relationship between the node and point (1..1 and 1…*) indicates that a node can be mapped to at least one point. This implies that points can overlap. Similarly, the representations allow faces defining surfaces to overlap (1…* and 0..*).

4. Hybrid suitability

Like the 3D FDS, the representation explicitly defines the features body, surface, line and point, making it suitable in this regard for integration with distance field representations. The DTM and texture features are not well defined in the representation. The lack of arcs in the line feature definition poses challenges to the definition of complex line features.

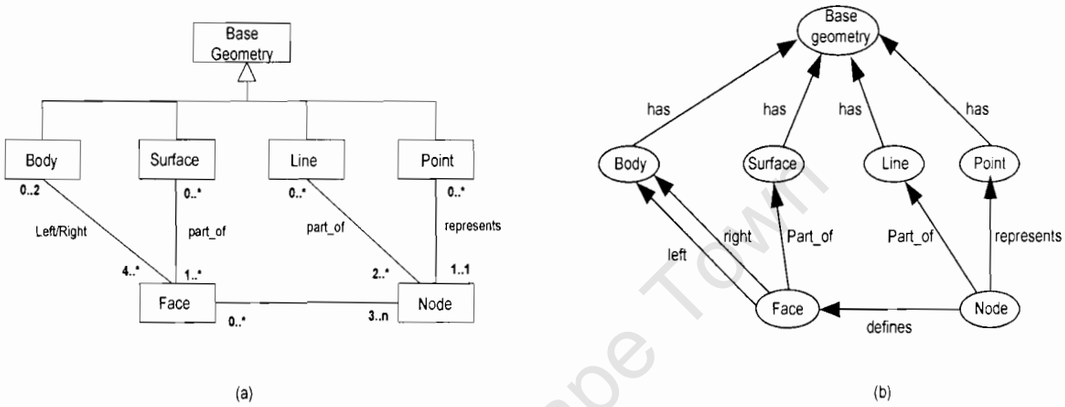


Figure 5.2. (a) Un-normalised form, Urban Data Model (UDM) (adapted from Coors, 2003) (b) UDM presented in flow-chart notation, for analysis purpose.

5 Spatial Analysis

The representation cannot directly model ***T1-T3***, ***T8*** and ***T9*** queries. This can easily be attributed to the lack of an arc primitive in the structure. The queries that could be modelled constitute 50% of the sample set.

Simplified Spatial Model (SSM): The SSM(Figure 5.3) was developed with the main trust towards visualisation in VR environments (see Zlatonova,2000). The summarised properties are given below:

1. Space complexity

The storage cost for the SSM representation is estimated to scale as $\boldsymbol{O}(N^{0.3089})$ (see Chapter 7). Since the SSM lacks the arc primitive, the space costs are expected to be this low,

compared with other representations. The SSM's space drawback comes from the fact that it is intended for visualisation and to some extent for animation purposes as it stores an objects rendering attributes (see Zlatanova, 2000). The other potential source of memory requirements comes from the fact that, if there are lines or points on a face, the face has to be divided into several sub-faces.

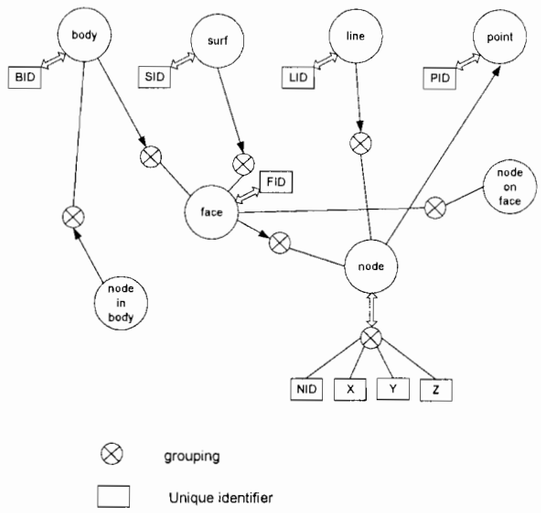


Figure 5.3. Simplified Spatial Model (SSM) (Zlatanova,2000)

2. Time complexity (efficiency)

For Queries ***T4-T7*** and ***T10***, the SSM is relatively, the most efficient representation (see Chapter 7). This high performance can be attributed to the fact that there are fewer tables to traverse compared to other representations. Notably, the arc attribute would have introduced a new table and thus expected the query time for ***T4*** and ***T7***.

3. Object representation

The representation defines four abstract geometric objects (features) namely point, line, surface and body. The thrust for the SSM is towards visualisation. The model only permits the existence of convex faces (Shi et al., 2002). It is not clear as to why the face-in-body singularity has to be modelled. This would then point towards an attempt at modelling the internal structure of the object where you have planes inside bodies. But it should nevertheless be noted that the initial approach taken for this representation is that of B-rep.

4. Hybrid suitability

The body feature is represented implicitly as a group of faces, a surface feature implicitly as a group of faces, and a line feature as a group of nodes. This would imply querying the database for constituent elements every time a feature object is requested, making the rep unsuitable for integration into a hybrid structure. Again, the fact that the arc primitive is lacking implies that the representation is not suitable for dynamically segmented utilities (see Bhunu, 1999).

5. Spatial Analysis

Similar to the UDM, the SSM lacks the arc primitive and as a result only the ***T4-T7*** and ***T10*** queries can be implemented directly. This translates to 50% of the sample set.

5.1.2 Discussion:

The fact that the SSM and UDM lack the line or arc primitive has not clearly indicated the gain in doing so. Instead, the two representations cannot be used for utilities modelling because of this lack of explicit definition of the arc primitive. Although the UDM and SSM representations have minimal gains in storage costs and efficiency over the 3D-FDS and proposed H3D in (see Chapter 7), the non-availability of the arc primitive means that they can only attempt approximately 50% of the spatial queries for the sample set ***T1-T10***.

The 3D FDS representation has the edge primitive solely for orientation. This information (orientation) can easily be derived from the arcs, hence rendering the edge primitive redundant. The node-in-body singularity has no practical application in urban design as such; it will increase the complexity of the representation. More space will be required to maintain relations caused by this link. The face "left" and face "right" relations will mean that a double traversal through the face records to establish relationships between face-based features (i.e. surfaces and bodies)

Generally, the UDM, SSM, Winged edge and 3D-FDS still underline the fact that there is no complete representation for urban 3-D modelling. This chapter will provide a B-rep component that addresses the limitations of these representations and at the same time incorporates sound theoretical design principles.

5.2 Conceptual design of the Hybrid 3-Dimensional representation (H3-D)

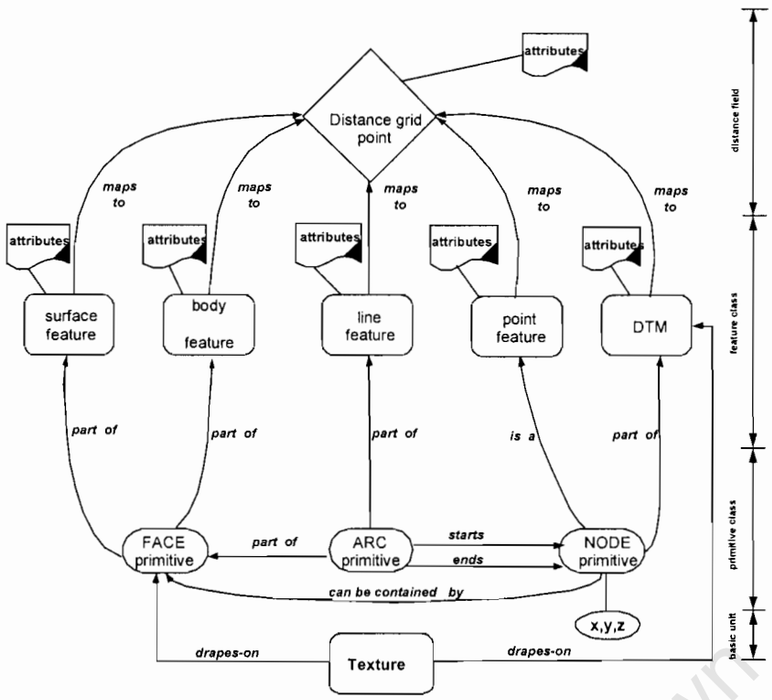


Figure 5.4. Conceptual design of the H3-D representation.

In this section, the definition of the new 3-D spatial representation will be provided. All components of the H3-D rep are defined in set theory notation. The formal definition of the data structure will establish the rules by which an object can be composed. Furthermore, we assume that all objects are embedded in Euclidean 3-D, $\boldsymbol{R}^3$.

The conceptual representation shown in Figure 5.4 above, can be fully explained by breaking it up into five sections;

Note: Let $\boldsymbol{U}$ be the world co-ordinate space and $\boldsymbol{p}$ any point within it.

5.2.1 Basic unit-the point:

This is the basic construct in H3-D rep modelling and it is a point in $\boldsymbol{R}^3$ space with a coordinate tuple (x,y,z). *Primary primitives* in H3-D are derived from this basic unit.

5.2.2 Primary primitives

The primary primitives are the smallest building blocks for H3-D defined features.

Node: When the point basic unit is put into the context of H3-D modelling, it is regarded as a node.

N_p Denotes an indexed set of Nodes of which node p_i is an element

$p_i \in N_p$ Where i is the unique index of a node (the index provides a unique identification of the node).

Properties: two nodes cannot be identical, i.e. they are always disjoint:

Arc: an arc can be defined as a straight-line segment with a node at either end. If ***TS*** is the topological space, then ***ND*** denotes the set of all the ***n*** nodes within the topological space. Two nodes $\boldsymbol{p}_a \in \boldsymbol{N}_i$ and $\boldsymbol{p}_b \in \boldsymbol{N}_j$ in the same topological space are connected if there is a straight line (or link) between them. This defines an arc ***a***. And arc ***a*** is an element of an indexed arc set $\boldsymbol{A}_i$ i.e. $\boldsymbol{a} \in \boldsymbol{A}_i$

The nodes that define the arcs are ordered, hence this results in arcs being directed. A constraint to be observed is that all arcs within the same line feature will have the same direction. The orientation on arcs is necessary for two primary reasons; Neighbourhood information relating to "right" and "left" can easily be obtained. Secondly, algorithms in some computer graphics hardware will require directedness in order to render faces correctly. Since arcs define the boundary of a face, they hence provide the orientation, which ensures correct rendering.

Face: The face primary primitive is made up of nodes and bounding arcs. A set of arcs A_j in the same topological space is denoted by $\boldsymbol{AT}$.

$$\boldsymbol{AT} \subset \boldsymbol{A}_i$$

A face f, is defined as an element of the indexed face set F_j: In B-reps, the face geometry is defined by its boundary, which can be expressed as an aggregation of arcs i.e.

$\partial f = \{a_1, a_2, \text{-------}, a_n\}$, for an n-sided face . Where ∂ defines the boundary.

Properties:

1. a face f will be defined by at least x number of arcs where $3 \leq x$, (for $x=3$, a triangular face is defined)
2. Two identical nodes cannot exist in the set.
3. Each set of three nodes defining the face, $\{ p_a \}$, $\{ p_b \}$and $\{ p_c \}$ fulfils only one planar equation
   $ax + by + cz = 0$ in R^3, i.e. the face is planar.

5.2.3 Features

The B-rep component of the representation uses a combination of the above three primitives to construct four feature objects, namely *point, line, surface* and *body*.

Point feature: A point feature is a special node, which, on its own, can model a world object, for example a man-hole, specific location, and all those features that are best described by assigning attributes to points. A point feature denoted by P_k^T is an element of an indexed set of points N_e

i.e. $P_k^T \in N_e$ and $N_e \subset N_i$ where T indexes attributes of the nodes also $(N_i - N_e) \cap N_i \neq 0$, therefore there are nodes which do not constitute point features.

Line feature: this models a linear object, for example water pipes, electricity lines and any other linear objects. A simple line feature is made up of one arc with attributes attached and a number of arcs put together will give rise to a complex linear feature.

A line denoted by L_k^T belongs to an indexed set of lines LN_t and is also defined as an aggregation of A_i^l arcs which are in the same topological space;

$L_k^T = \bigcup_{i=1}^{n} A_i^l$ Where T is the attribute index for a line, l is the line index and $1 \leq n$

And when $n = 1$, defines a simple line with a single arc.

All the arcs that take part in the generation of lines belong to the set AL i.e.

$AL = \{ A_i^l \}$

And $(A_i - AL) \cap A_i \neq 0$, i.e. there are arcs which do not constitute lines.

A line embedded in R^3 is therefore represented as a finite sequence of arcs. A line must have at least two points that are disconnected and no two nodes are equal (i.e. a line must not be self intersecting). This is of high significance in urban design as all the intersections of linear features represent important events (e.g. roads intersection, hence need attribute values to be attached), indicate the beginning of new feature objects and by prohibiting self-intersection, the representation ensures that non-contradictory topological information is properly stored.

Surface feature: A surface feature is an aggregation of faces with attributes attached. The surface feature can model patches of special importance. Continuous walls, roofs and ground surfaces are some of the objects that can easily be modelled by this feature.

A surface denoted by S_k^T is an element of an indexed set of surfaces and the surface is an aggregation of faces F_j^s i.e.

$S_k^T = \bigcup_{1}^{n} F_j^s$ where s is the surface index of the face and T is the attribute index for the surface

All the faces which constitute surfaces belong to the set FT ; $FT = \{ F_j^s \}$

And $(F_i - FT) \cap FT \neq 0$ i.e. there are faces which do not constitute surfaces

Surfaces are homeomorphic to 2-manifold, i.e. they are simple closed boundaries.

Body feature: The body feature is defined by a number of bounding faces. A body denoted by D_k^T is an indexed element of a set of bodies and aggregation of faces in the same topological space ***DF.***

$D_k^T = \bigcup_4^n F_j^d$ where ***d*** is the body index, **T** is the body attribute index. If ***n=4*** then a simple tetrahedron is defined.

And $DF \subset F_i$, $(F_i - DF) \cap F_i \neq 0$ i.e. there are faces that do not constitute bodies.

Special feature-Grid point: The grid point is part of the distance map. It is the reference from which distances are measured. To each point, a number of distance fields relating to different objects can be encoded as attributes of the point (see Figure 5.5, below), illustrating the *multi-distance fields* concept.

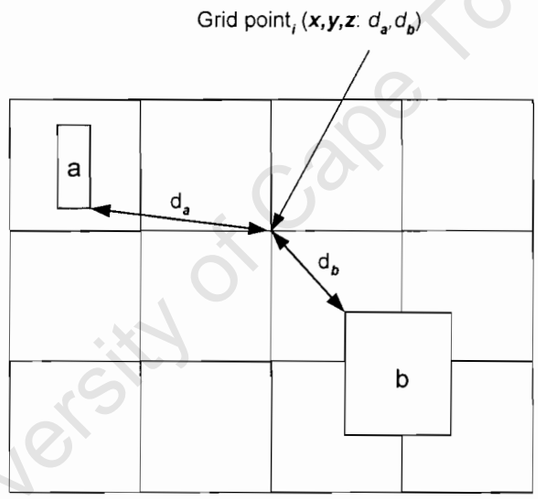


Figure 5.5. An illustration of two objects being represented in a distance field. Distances are radiating to the different corners of the objects

Special feature-Digital Terrain Model (DTM): The digital terrain model is created from the nodes and a particular algorithm is used to generate it. There are different approaches to the generation of DTMs and reference is made to a multitude of

literature in Geo-Information Systems, Photogrammetry and Remote Sensing (Laurini et al.1992)

Special feature-Texture: Texture is recorded for surfaces as raster data and applied during visualisation. The data representation allows the user to experiment with different texturing. Research into texture mapping has been well covered in computer graphics and issues related to this area are only mentioned at this conceptual level .

Interconnections: Table 5.2, below provides a summary of how the features making up the representation are interconnected.

Link	Features	Explanation
Starts , Ends	Arc Primitive / Node Primitive	An arc is defined by a beginning and an end node.
Part-of	Arc Primitive / Face primitive	The face is made up bounding arcs, which are part-of a face.
Can be contained by	Node Primitive / Face Primitive	In some instances (termed singularies), a node might be contained by a face for the purpose of capturing information e.g. when a line "cuts" through a face, a node is the "entry" point.
Part-of	Face Primitive / surface feature	A surface feature is an aggregation of faces, a face might be part of a surface
Part-of	Face primitive / body feature	A Body feature is made of a number of bounding faces, hence a face could be part of this body.
Part-of	Arc-Primitive / line feature	A line feature is made up of at least one arc, so an arc could be part of line feature.
Is a	Node Primitive / point feature	If a node primitive is used to represent a point object, e.g. manhole, air mark etc, it is transformed to a point feature
Part-of	Node Primitive / DTM	Using relevant DTM algorithms, the node is used to generate the required Digital elevation model.
Maps to	Surface feature/ Grid point; Body feature/ Grid point; Line feature/Grid point; Point feature/Grid point DTM feature/ Grid point	The link indicates a mapping using the VTK tool to produce a distance field map of all the features. A grid point, in the distance map will have as its attributes distances to all the features recorded in the map.
Drapes on	Face primitive/ Texture	Indicates the fact that a face might have texture draped on it.
Drapes on	DTM/ Texture	A DTM might have some texture applied onto it.

Table 5.2. Explanation of Links between building components of the H3-D rep.

The fusion between the boundary representation and Distance Field Modelling: The generated feature types of the boundary rep provide the link to the distance map. The

mapping operation involves reading the feature geometry(vector format) into distance field transform algorithms (see Chapter 3 and Chapter 6) and in the process, converting the feature into a distance map. The single Distance Field(sDF), a distance field modelling variant (see section 5.4.1.2) is integrated with the B-rep to form the H3-D representation. A justification for the use of sDF is provided in section 5.5.4.

5.3 Formalism of the H3-D rep: Boundary rep component.

The formalism to the H3-D is the definition of the constraints that have to be adhered to in aggregation of the primitives as they make up the 3-D models and the definition of relations and *cardinalities* in database design. This section outlines the potential of the representation to implement spatial queries. Topological relations have been used in GIS communities as the most appropriate manner for describing spatial relations. Hence the thrust will be towards estimating the ability of the representation to identify topological relations. The derivations of topological relations are based on the framework provided by the 9-intersection model (see Engehofer and Herring 1992). Zlatanova (2000) makes a comprehensive study of possible, redundant and negative topological relations between two identifiable objects. This research will be limited to the illustration of the possible topological relations that can be implemented by the representation. The matrix below provides the description of a disjoint topological relationship between two objects using the 9-intersectionmodel.

$$\boldsymbol{R}(\boldsymbol{A},\boldsymbol{B}) = \begin{pmatrix} \boldsymbol{A}^{o} \cap \boldsymbol{B}^{o} & \boldsymbol{A}^{o} \cap \partial\boldsymbol{B} & \boldsymbol{A}^{o} \cap \boldsymbol{B}^{-} \\ \partial\boldsymbol{A} \cap \boldsymbol{B}^{o} & \partial\boldsymbol{A} \cap \partial\boldsymbol{B} & \partial\boldsymbol{A} \cap \boldsymbol{B}^{-} \\ \boldsymbol{A}^{-} \cap \boldsymbol{B}^{o} & \boldsymbol{A}^{-} \cap \partial\boldsymbol{B} & \boldsymbol{A}^{-} \cap \boldsymbol{B}^{-} \end{pmatrix} = \begin{pmatrix} 0 & 0 & \neg 0 \\ 0 & 0 & \neg 0 \\ \neg 0 & \neg 0 & \neg 0 \end{pmatrix}$$

where $\boldsymbol{A}^{o}$, $\boldsymbol{B}^{o}$ are internals for objects $\boldsymbol{A}$ and $\boldsymbol{B}$

$\partial\boldsymbol{A}, \partial\boldsymbol{B}$ are the boundaries of objects $\boldsymbol{A}$ and $\boldsymbol{B}$

$\boldsymbol{A}^{-}, \boldsymbol{B}^{-}$ are the externals of objects A and B

$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$ are the closures of objects **A** and **B**

$\boldsymbol{A}^{o}, \partial\boldsymbol{A}$ and $\boldsymbol{A}^{-}$ are the topological primitives of an object $\boldsymbol{A}$.

5.3.1 Topological relations and constraints in H3-D modelling

This section illustrates the available topological relations and the constraints within the H3-D representation, incorporating an analysis on the B-rep component.

5.3.1.1 Point Relations

The point based relations are explained below:

Point versus point: The relationships for points is that of disjoint and coincidence. The disjoint relation (according to set-theory) requires that the two boundaries be disjoint i.e. $\partial A \neq \partial B$ for points A and B. The H3-D representation accepts that should there be two points coinciding then this will be accepted as an error in modelling. The corresponding disjoint relation can be implemented in the relational mapping of the representation by having different ***IDs*** for the objects.

Point Versus Line: The relationship between the point and line basically tests for equality.

Let p_n be the node defining the point object i.e. $\{N_p\} = p_n$, where N_p is indexed node set with one element p_n .And let the nodes defined within the line's closure set be L =$\{ p_1, p_2, ..., p_m \}$. Then the intersection test $N_p \cap \bigcup_1^m p_i \neq 0$ for some $p_i = p_n$, indicating the existence of intersection.

Point Versus Surface: The point-surface/face relationship is one of the most important in urban design. It helps in assigning attribute features to points on the wall of building or in defining the entry point of a linear feature through a wall, thus enabling the definition of the relationship between the wall and the feature (see Figure 5.6 below).

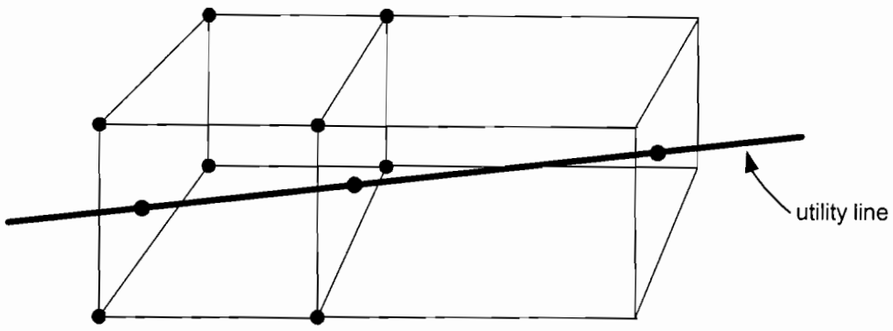


Figure 5.6. Relationship between a linear feature and walls of a building feature

Let $\boldsymbol{p}_n$ be the node defining the point object, i.e. $\{\boldsymbol{N}_p\} = \boldsymbol{p}_n$, $\boldsymbol{N}_p$ in the indexed set node.

And let the nodes defined within the faces closure set be $\boldsymbol{F} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_m\}$. The assumption: a face contains at least an internal node is made, otherwise the relation would fail to materialise.

Then $\boldsymbol{N}_p \cap \bigcup_{1}^{m} \boldsymbol{N}_i \neq 0$ for some $N_i = N_p$ indicates the existence of some intersection between the walls and linear object. Otherwise, intersection will not exist.

Point Versus Body: This relationship is a singularity in 3-D modelling (i.e. unique occurrence) (Figure 5.7, below). Although this singularity is mentioned here, it does not have any practical applications in urban modelling. All urban features are linked to one another somehow. As such, the H3-D representation's implementation will not capture this singularity.

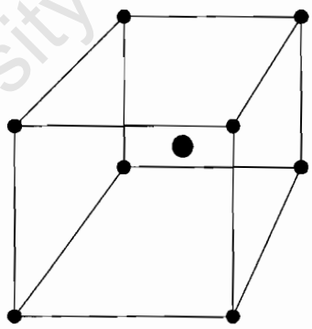


Figure 5.7. Node in volume defines a node that is included in a volume.

5.3.1.2 Line relations:

Line versus Line: The line relations are intersection, disjoin and meet. In H3-D modelling, arcs cannot have non-empty intersections (using the set theory). Where arcs pass through the same plane and meet, a node should be introduced (Figure 5.8, below).

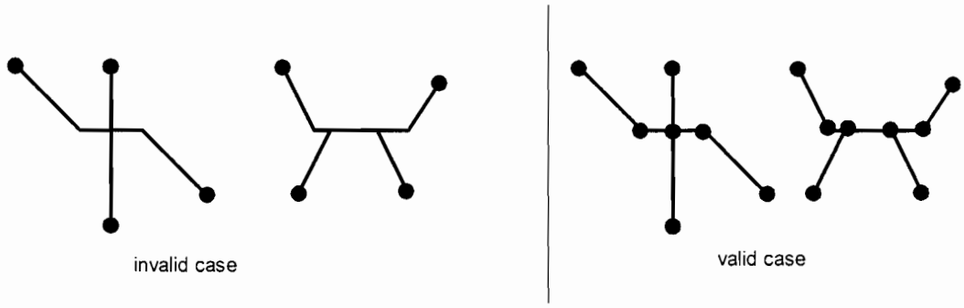


Figure 5.8. Illustrating arc versus arc formalism

Meet relation: This relation (see Figure 5.9, below) addresses the issue of line features meeting through the intersection of their boundaries. Their internals are therefore not investigated. The analysis process presupposes that one of the two bounding nodes of ***A*** has to coincide with one of the bounding nodes of ***B***. No other common nodes are allowed, i.e.

Let the bounding nodes set for ***A*** be $\partial N^a = \{p_{a1}, p_{a2}\}$ and for ***B***, $\partial N^b = \{p_{b1}, p_{b2}\}$

Where $\partial N^a \subset \overline{A}$ and $\partial N^b \subset \overline{B}$

Then $\{\overline{A} - \partial N^a\} \cap \{\overline{B} - \partial N^b\} = 0$, i.e. the line features do not intersect using their internals in this instance.

And $\partial N^a \cap \partial N^a \neq 0$ i.e. the two lines meet.

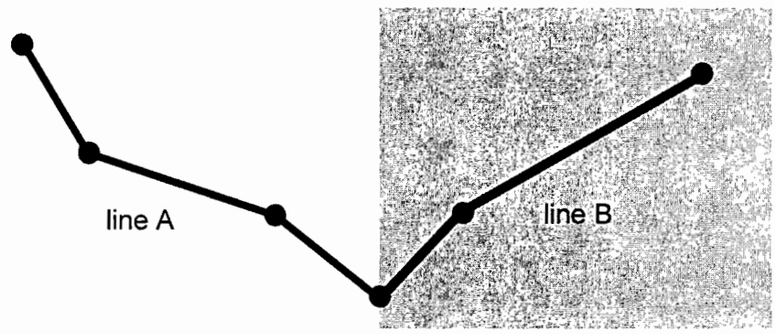


Figure 5.9 Two line features illustrating the meet relation in H3-D

Intersection relation: Intersection can be defined and illustrated by the diagram below (see Figure 5.10). The lines can both intersect through their internals (Figure 5.10b), or one of them can intersect through its internal and the other, through the bounding node (see Figure 5.10a)

Let the Closures for lines A and B include their bounding and internal nodes i.e.

$\overline{A} = (\{\partial N^a\} + \{(N^a)^o\})$ and $\overline{B} = (\{\partial N^b\} + \{(N^b)^o\})$

Then $\overline{A} \cap \overline{B} \neq 0$ hence this implies that the intersection relation can imply the meet relation but the reverse is not true.

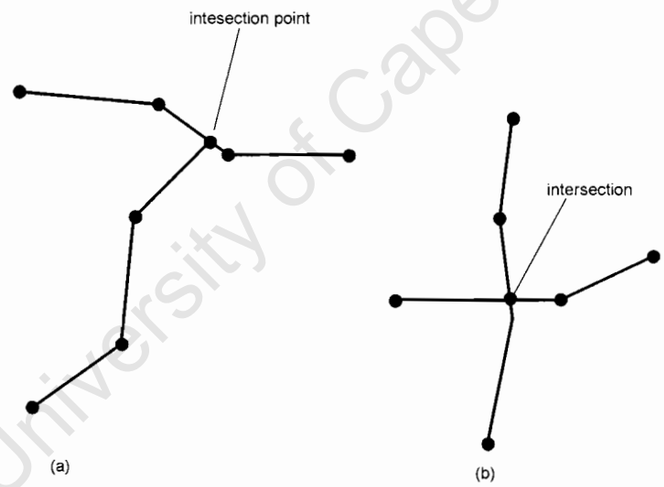


Figure 5.10. Intersections. Diagram(a)showing Boundary-internal node intersection and diagram(b) showing internal-internal node intersection

Disjoint relation: This relation can be detected by the Boolean analysis of common nodes that belong to the closure of A and the closure of B i.e.

if $\{N_1,.....,N_k\} = \overline{A}$ and *if* $\{N_1,.....,N_m\} = \overline{B}$,

then $\bigcup_{i=1}^{k} N_i \cap \bigcup_{j=1}^{m} N_j = 0$ implies disjoint relationship.

Line Versus Surface relations

Case 1: line lying on the surface

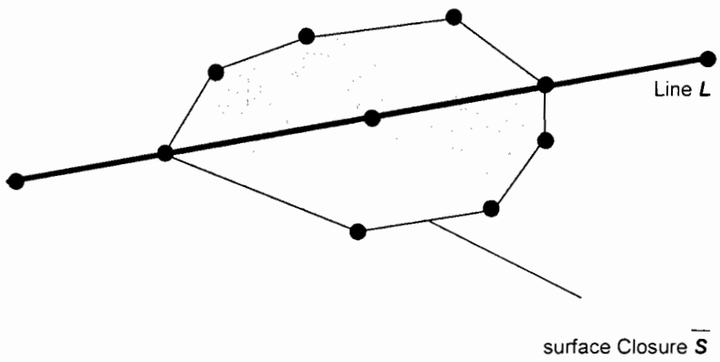


Figure 5.11.Line versus Surface relation

The surface closure defined by $\overline{S} = \partial S + S^o$, the boundary ∂S defined by the set of surface nodes N_i^S and line L by $\overline{L} = \partial L + L^o$

The above relationship (Figure 5.11) is only possible if we have common nodes between the boundary of the surface and the internal of the line i.e.

$\partial S \cap L^o = inN \neq 0$ and $inN \subset L^o$, $inN \subset \partial S$ where inN is the intersection set.

Without internal nodes relations, the line will not "touch" the internal surface, but "fly" over it.

Case 2: Line touching surface boundary

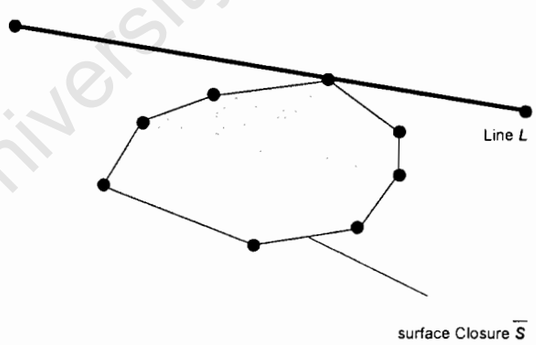


Figure 5.12. line "touching" a surface

if the line touches the surface of a single point (Figure 5.12, above), then:

number of elements in the set $n(inN) = 1$

Case 3: A Line going through the surface (piecing the internal of the surface), see Figure 5.13, below.
This relationship is only possible when the surface has internal nodes not in the same plane as that of the line i.e.

$S^o \neq 0$ and the following inference can be made:

$S^o \cap L^o \neq 0$ and $S^o \cap \partial L = 0$ implies that the line and surface internals only are intersecting and this illustrated diagram Figure 5.13 below (valid case)

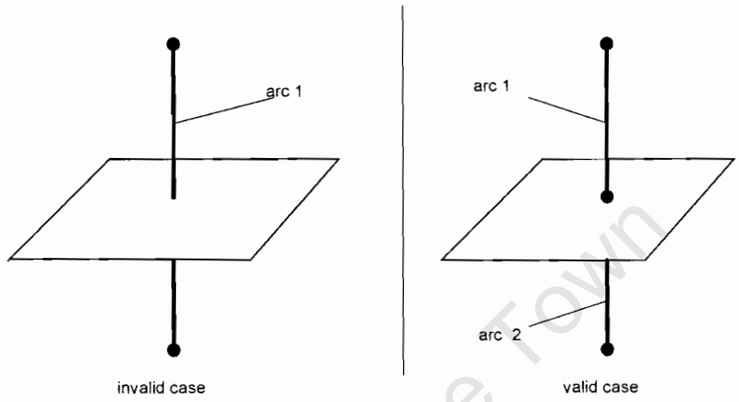


Figure 5.13. Intersection between an arc and a face

5.3.1.3 Surface Relations:

Case 1: A disjoint B
If $\{F_1,.....,F_k\} = \overline{A}$ and $\{F_1,.....,F_m\} = \overline{B}$ then the two surfaces are completely disjoint when $\bigcup_{i=1}^{k} F_i \cap \bigcup_{j=1}^{m} F_j = 0$. However this does not give a complete solution to all possible scenarios as we can have the internal of a face intersecting with the boundary edge of another face (see Figure 5.14, below). Hence, a comprehensive approach would be to make the analysis using the closures (making use of the nodes).

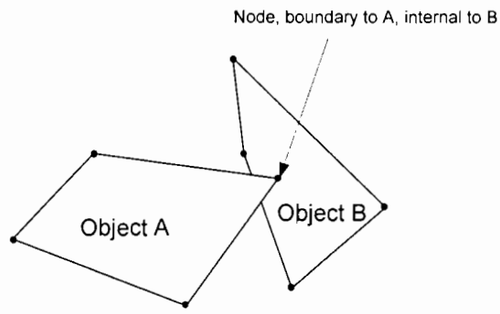


Figure 5.14. Internal of a surface intersecting with the boundary of another

So the analysis: $\overline{A} \cap \overline{B} = 0$ for disjoint objects, implies that the internals and boundaries are disjoint as well.

Case 2: ***A*** inside ***B***

If $\{F_1,.....,F_k\} = \overline{A}$ and $\{F_1,.....,F_m\} = \overline{B}$ Then $\bigcup_{i=1}^{k} F_i \cap \bigcup_{j=1}^{m} F_m \neq 0$

And if B is wholly contained by A,

Then $\overline{B} \subset A^o$

Case 3: ***A*** meet ***B***. The intersection of two faces results in an arc being formed along the line of intersection. This relation requires the analysis of the boundary and internal nodes. Figure 5.12 below illustrates a surface boundary nodes and internal nodes of the other defining the intersection of the two. In some cases, surfaces can meet and intersect boundary nodes (see Figure 5.15, below).

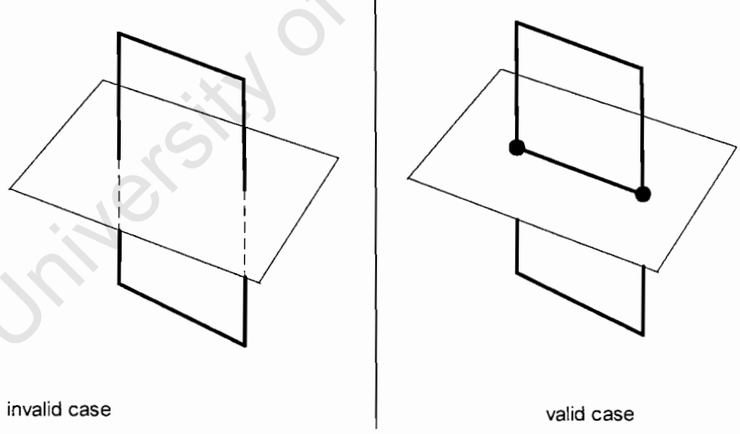


Figure 5.15. Intersection between two faces. Internal and boundary nodes intersection.

When surface A meets B

$\overline{A} \cap \overline{B} \neq 0$ i.e. meeting the boundary and internals (see Figure 5.13 below)

if B intersects with the internal of A, then the following result is obtained: $A^o \cap \overline{B} \neq 0$

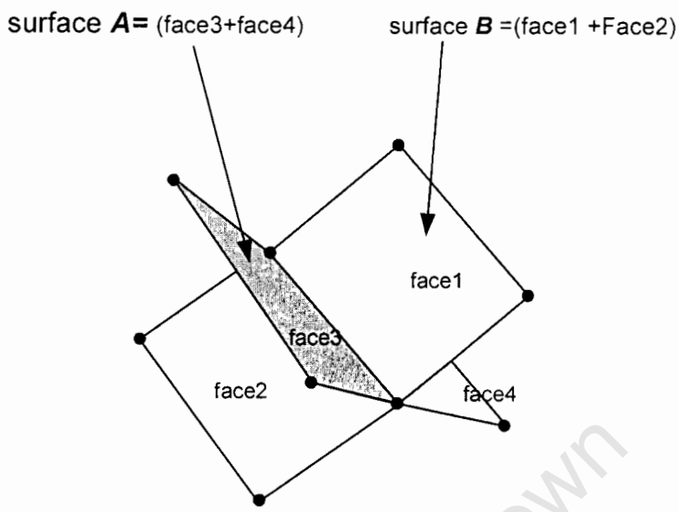


Figure 5.13: Surfaces meeting through their boundary and internals

5.3.1.4 Body Relations

Bodies are a special model that is composed of faces enclosing space. The boundary of the body is the set of faces and the interior of a body is either space, faces or nodes. According to the H3-D, the urban domain will not make use of a modelling approach in which the interior of a body will have "suspended" face or point features. Theoretically this is possible, but this research finds no practical application. All primitive features are connected or "attached" to the main body elements. For example, a "hanging" electric bulb is attached or connected to the ceiling or roof through an electric cable. Therefore, most of the body relations are derived from those already defined for Lines and Surfaces, logically so since the body feature is built from these features.

Case 1: Body A disjoint from Body B

When two bodies are perceived to be disjoint, some investigation through the face primitives is carried out.

The closure test i.e. $\overline{A} \cap \overline{B} = 0$ normally provides a conclusive result to the fact that A and B are disjoint. In some cases, the test on boundaries will provide the desired results, assuming that one of the bodies is not contained entirely in the other body, which would mean that the test fails to detect:

$\partial A \cap \partial B = 0$ i.e. Bodies A, B disjoint as long as their internals are disjoint.

Case 2: A meets B

The meet relation between two bodies can be achieved through the "touching" of faces, nodes or edges (see Figure 5.17, below)

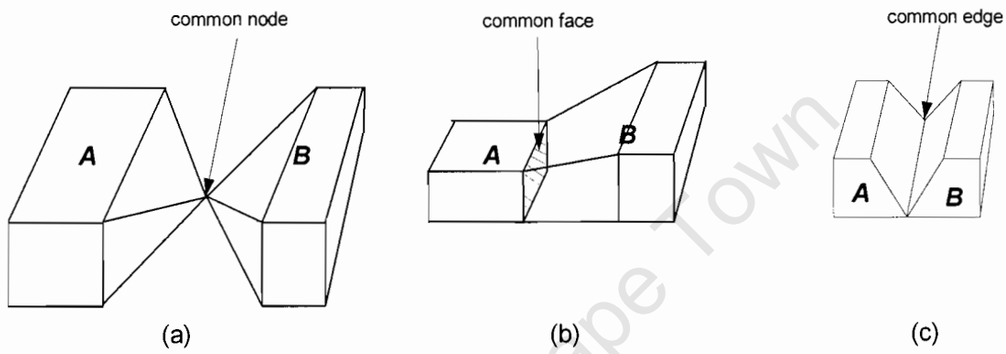


Figure 5.17. Illustration of the meet relation within body features

In situation ***(a)*** above, the relationship can be defined at the level of boundary node investigation i.e. $\partial N^a \cap \partial N^b \neq 0$. Situation ***(b)*** can be well defined by an analysis of the body faces . If $\partial A = \{ F_1, \ldots\ldots, F_k \}$ and $\partial B = \{ F_1, \ldots\ldots, F_m \}$ then $\bigcup_{i=1}^{k} F_i \cap \bigcup_{j=1}^{m} F_m \neq 0$. When the intersection is along the arcs, the case can simply be treated as situation ***(a)***. The only condition is : $\left\| (\partial N^a \cap \partial N^b) \neq 0 \right\|$ i.e. the number of nodes is at least 2.

Case 3: A overlaps into B

The intersection of two bodies will result in the formation of a face in the plane of intersection (see Figure 5.18, below).

The relationship can be defined by:

$\partial N^B \cap \overline{A} \neq 0$, the boundary intersection of B intersecting with the closure of A, indicates that object $\boldsymbol{A}$ at least touches $\boldsymbol{B}$ and $(N^B)^o \cap \overline{A} \neq 0$ indicates that some or all of A is contained in B.

The combination $\partial N^B \cap \overline{A} \neq 0$ and $(N^B)^o \cap \overline{A} \neq 0$ will fully define the overlap relationship.

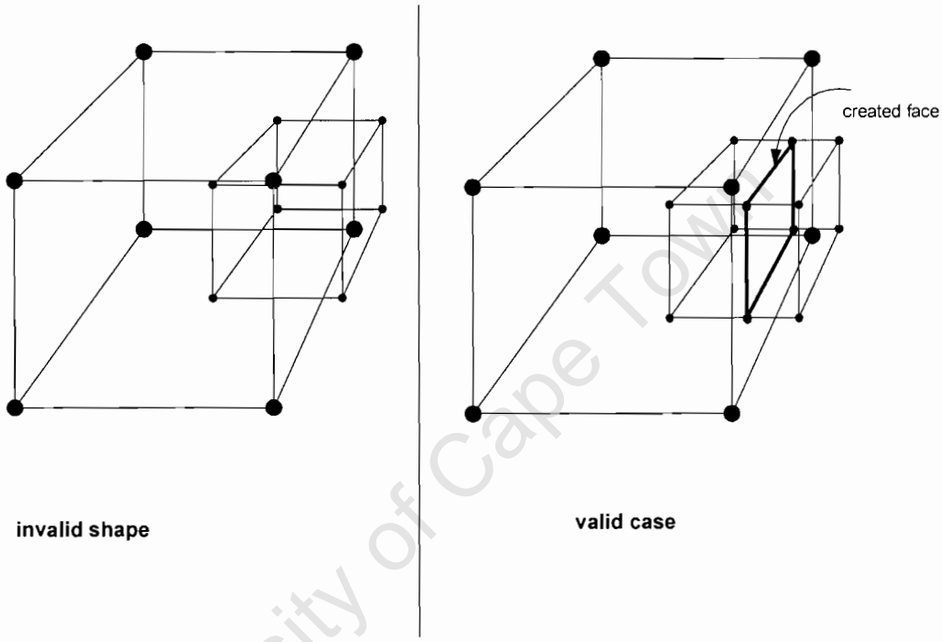


Figure 5.18. Intersection between two bodies

Case 4 $\boldsymbol{A}$ inside $\boldsymbol{B}$: For this relationship to be investigated, the internal of B must not be empty i.e. $\boldsymbol{B}^o \neq 0$ and then $\boldsymbol{B}^o \cap \overline{\boldsymbol{A}} \neq 0$. To ensure full containment, $\partial N^B \cap \overline{A} = 0$ which means that object $\boldsymbol{A}$ does not cross the boundary of $\boldsymbol{B}$.

5.4 Explaining relationships and cardinalities- Entity Relationship Modelling

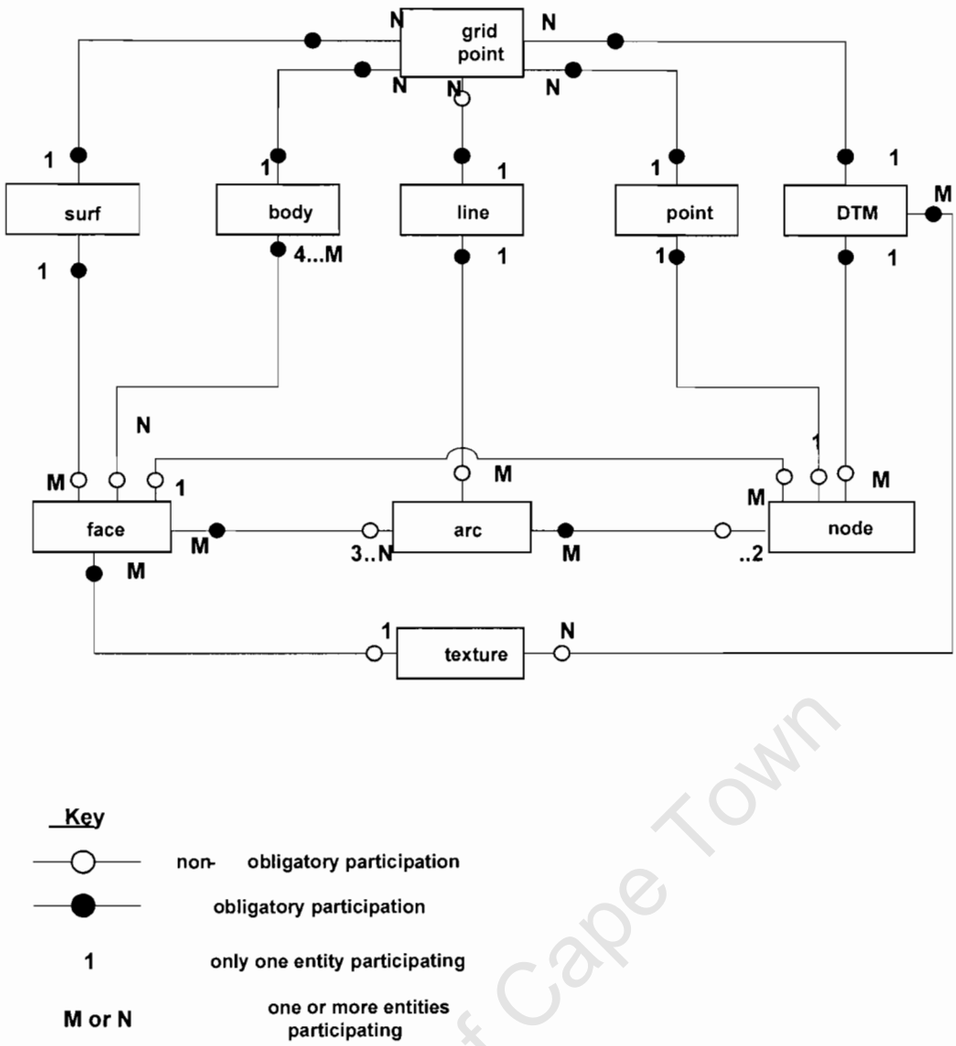


Figure 5.19 :Conceptual entity-relation model of the H3-D rep.

In general, the conceptual data structure presented in Figure 5.4, can be translated into different kinds of database structures. Choosing the relational database approach offers practical advantages, such as the ease of defining relations (tables) and availability of mature database management systems (Howe, 1989). The set theory relations are also easily implemented in a relational database. Figure 5.19 above, provides the mapping of the conceptual design to an Entity-Relational data model.

The enterprise rules, which are necessary in defining the relationships, are given in detail in Appendix *A*. Figure 5.20 is an illustration of the logical design (E-R model), which is a result of the application of modelling rules to the conceptual Entity-Relationship model (Figure 5.19). Details of Entity Relationship modelling are well covered by Howe (1989).

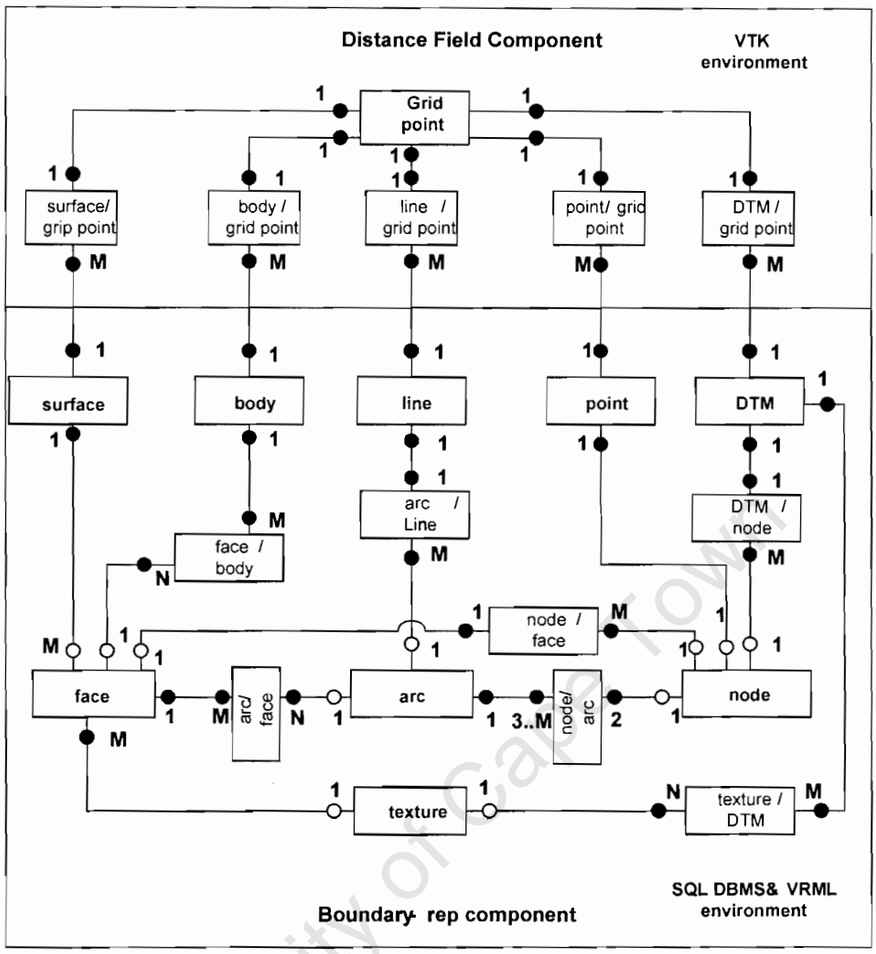


Figure 5.20. Logical design for the H3-D representation.
Note: the model applies for the *sDFs*.

Explaining the E-R notation:

- Non-obligatory participation (blank circle): The entity type (class) is physically represented by a table made up of a number of members (or entities) . The non-obligatory statement implies that not all members of the entity type will participate in the relationship modelled with another entity type.

- Obligatory participation: This basically means that all members (or entities) will participate in the modelled relationship.
- ***M*** or ***N***: This notation (where there is an ***M*** or ***N***) is an indication of the fact that at least one member will participate in the relationship. The ***M*** and ***N*** have the same status, they are used in a many to many relationship to differentiate between the two groups of entities participating in the relationship. In an case of one to many, only the M symbol is used.
- One (1): Emphasises that only one member is expected to participate in a relationship.

Logical Model: The logical model is a result of the application of E-R modelling rules (see appendix ***A***) to the H3-D entity-relationship conceptual model.

Earlier sections in this chapter propose the use of set theory to explain the possible relations that are implemented by the H3-D rep. The sections below, provide a detailed outline of how mapped set theory relations are implemented through the E-R modelling approach. This emphasises the need to clearly define all the relations used in set theory so that the E-R modelling approach can easy translate the mappings.

5.4.1 Grid Point to Point/DTM/Line/Body or Surface Relation. (Cardinality: Many to One (*M:1*))

This relationship is obligatory on the two entity types. The relationship implies that a number of distance field values (which are referenced on the grid point) can be used to define the Point (or DTM, line, body and surface). At the same time, the representation (i.e. H3-D) makes it clear that the field point contains only one value (for the ***sDF***).

5.4.2 Node to Arc Relation. (Cardinality: Two to Many (*2:N*))

Obligatory on the arc entity and non-obligatory on the node entity. The relationship points to the fact that nodes are used in defining arcs, and a node could be common to a number of arcs. The import spatial operation that benefits from this relationship is that of adjacency. Through the common node, adjacent features can be defined (see

Figure 5.21, below). The fact that the relationship is non-obligatory on the node side, means that the node entity class contains nodes which are not all necessarily used for arc modelling. There is a maximum of two nodes per arc.

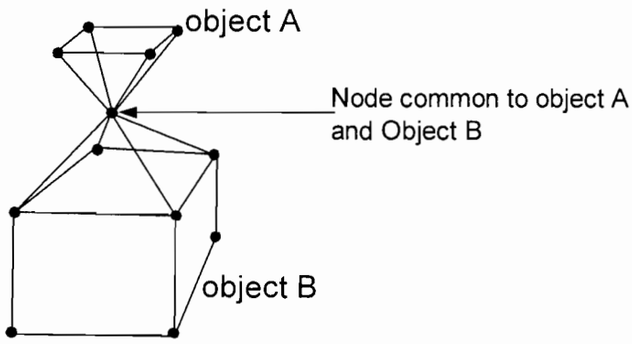


Figure 5.21. Showing two body features linking through the node to form a complex object.

5.4.3 Node to Face Relation. (Cardinality: Many to 1 (*M:1*))

Non obligatory on both sides. The fact that it is a non-obligatory relationship on both sides simply means that the relationship is quite rare in practice. This caters for one of the singularities in spatial modelling, where a node is contained in a face.

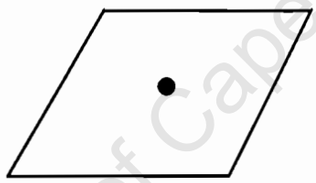


Figure 5.22. Defines a node that is included in a face.

The constraint to this singularity (Figure 5.22, above) is that the node cannot be an end-point of an arc that is included in a face or in an arc bounding a face. However, the node can link to an arc that is "cutting" through the plane of the face.

5.4.4 Arc to Face relation.(Cardinality: Many to Many (3..N,M))

Non-obligatory on the arc entity and obligatory on the face entity. The relationship illustrates the fact that many arcs can be used to define a face, and on the other hand, the arc can be common to a number of faces, helping to model complex objects

(Figure 5.23, Below). A minimum of 3 arcs is required to define a face. One of the strengths of the H3-D rep lies in complex object modelling. Manifold and non-manifold complex objects can be represented, i.e. an arc can be shared by more than two faces (non-manifold scenario).

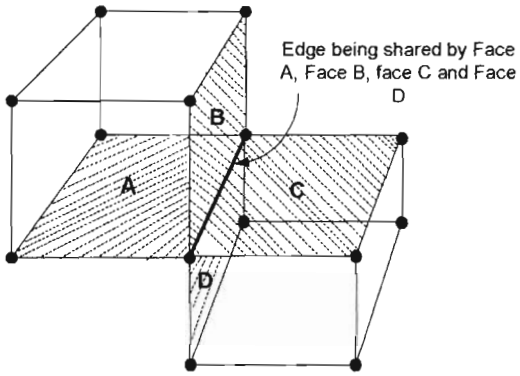


Figure 5.23a. A non-manifold object showing a common arc between cubes.

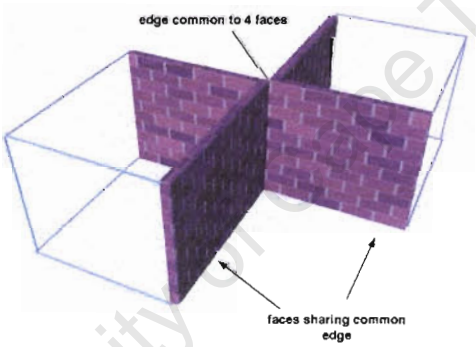


Figure 5.23b. Practical application. This could be related to a topological query of the form "Show the walls that are incident on this edge i.e. arc (id=y ?"

5.4.5 Arc to Line Relation. (Cardinality: Many to 1 (M,1))

Non-obligatory on the arc entity and obligatory on the Line entity. Once an arc is used to define a linear feature(e.g. a utility line), then it is recorded as a line feature. The relationship remains non-obligatory on the arc entity simply because not all arcs are used for modelling linear features.

5.4.6 Face to Surface relation. (Cardinality: Many to many (M,N))

Obligatory on the surface side, and non-obligatory on the face. In this model, faces aggregate to make up surfaces. The relationship has to be non-obligatory on the face

side because some faces participate in relationships with body objects without necessarily being converted to surfaces. Since the relationship is many to many, it allows for the face or surface to be shared between two objects. This enables for adjacency relationships to be derived. Figure 5.24, gives an illustration of a shared face between two body features.

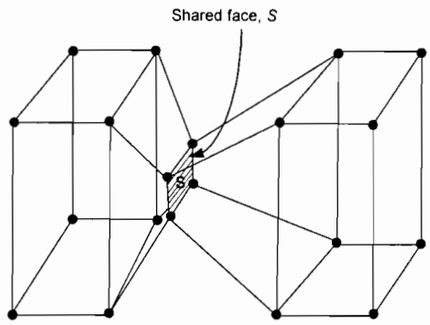


Figure 5.24. Sharing a face between two body features.

5.4.7 Body to face relation. (Cardinality: Many to many(M,N))

Obligatory on the body entity and non-obligatory on the face entity. The body-face relationship can be thought of as a decomposition of a body-body relationship. The link between two bodies is the common face between them. A body feature is made up of a number of faces and on the other hand, a face can be incident on two bodies. This is captured in the relationship by the fact that the cardinality is Many to Many. Again, the adjacency relationship between two body features can be deduced from this association.

5.4.8 Body to Line. (Cardinality: Many to Many (M,N))

Non obligatory on both sides of the entities. In urban environments, the containment operation is very important. A linear feature could be contained in two adjacent buildings and this has to be properly captured in a 3-D system. This relationship ensures that a line feature passing through a body is related to the faces (hence the body) that it "cuts"through. A linear feature can be found in many body features and many linear features can be located within a single body (Figure 5.25, below).

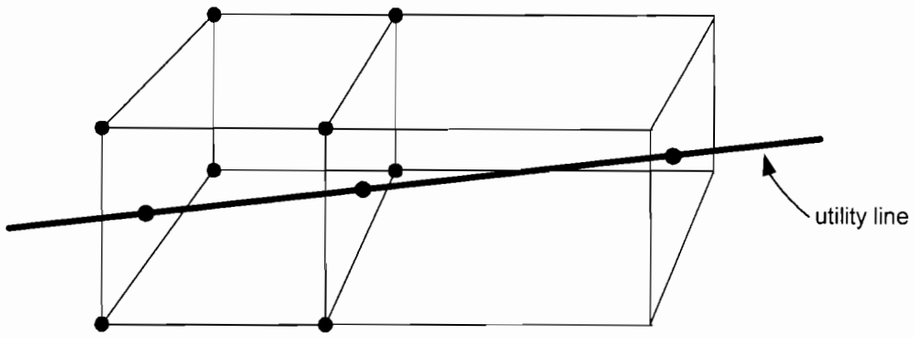


Figure 5.25: A utility line "cutting" though two body features. Note that nodes are introduced where the utility crosses a surface.

Note that this singularity (line in body) is not directly captured, but can be derived from the representation. That is, the node face relationship is defined, and two nodes on the two faces will have a linear feature joining them.

5.4.9 Body to Point relation.

Another singularity is that of point in Body (Figure 5.26, below). Although this singularity is mentioned here, it does not have any practical applications in urban modelling. All urban features are linked to one another somehow. The constraint to this singularity is that the node in a volume is isolated from arcs and is not included in a face.

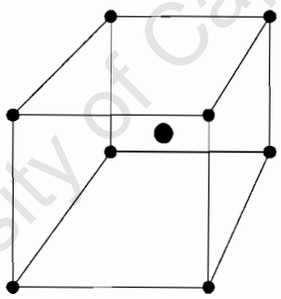


Figure 5.26. Node in volume defines a node that is included a volume.

5.5 Distance field modelling

This section provides two alternative distance field structures for modelling urban environments. A comparison based on space costs and spatial querying capabilities

will be carried out and the most appropriate option will be integrated with the B-rep component. Comparisons between B-reps and DFs will be made on some queries which have not been efficiently implemented in B-reps. In carrying out the spatial analysis operations, the distance field component thrives on making maximum use of distance field values, the derived gradient and trilinear interpolated values (see appendix C).

Set theory is applied to fully detail spatial modelling using distance fields (as is the case with B-reps). For any surface set defined in the distance field $\boldsymbol{S}$, a function $\boldsymbol{u}: \boldsymbol{E}^3 \rightarrow \boldsymbol{R}$ is a distance field function if it associates with any point $\boldsymbol{p}$ in space a real number equal to the accepted shortest distance (i.e. Euclidian, Manhattan, etc.) from $\boldsymbol{p}$ to $\boldsymbol{S}$ (Bradley *et al.* 1992).

The following properties of distance fields are important in spatial modelling:

1. a point $\boldsymbol{p}$ belongs to the set $\boldsymbol{S}$ if and only if $\boldsymbol{u}(\boldsymbol{p}) = 0$, which means that $\boldsymbol{p}$ is on the boundary of S i.e. $\boldsymbol{p} \in \partial \boldsymbol{S}$.
2. The distance function $\boldsymbol{u}$ is differentiable to zero (with respect to the three distance components along the axes $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{z}$) at those points that are equidistant from two or more surfaces.
3. Gradient estimation for spatial analysis purposes is derived by the central difference method and non-grid points scalar values are obtained through trilinear interpolation. Higher order functions do exist but they are deemed too expensive for our purposes (see Jones *et al.*1994)

   Using the central difference method, the gradient is $\boldsymbol{g}$ is computed from

$$\boldsymbol{g}_x = \frac{\boldsymbol{D}_{x+1,y,z} - \boldsymbol{D}_{x-1,y,z}}{2}$$

$$\boldsymbol{g}_y = \frac{\boldsymbol{D}_{x,y+1,z} - \boldsymbol{D}_{x,y-1,z}}{2}$$

$$\boldsymbol{g}_z = \frac{\boldsymbol{D}_{x,y,z+1} - \boldsymbol{D}_{x,y,z-1}}{2}$$

   where $\boldsymbol{g}_x, \boldsymbol{g}_y$ and $\boldsymbol{g}_z$ are gradient components in the $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{z}$ directions, respectively. $\boldsymbol{D}$ is the scalar value distance at a grid point.

5.5.1 The Distance Field representation design

The most important component of distance field modelling is the 3-D grid. Distance field modelling makes use of an adaptive grid for two major reasons; to define the "sharp" edges of an object and to conserve on space (see chapter 3).

The adaptive grid proposed in this research will be mapped through an octree hierarchical structure (Figure 5.27). The grid will have regions of varying cell resolution. In the hierarchical structure, each node will represent a grid cell, and will have the following information:

- Pointers to the parent and children (if it's not a leaf node)
- Distance values of the cell corners

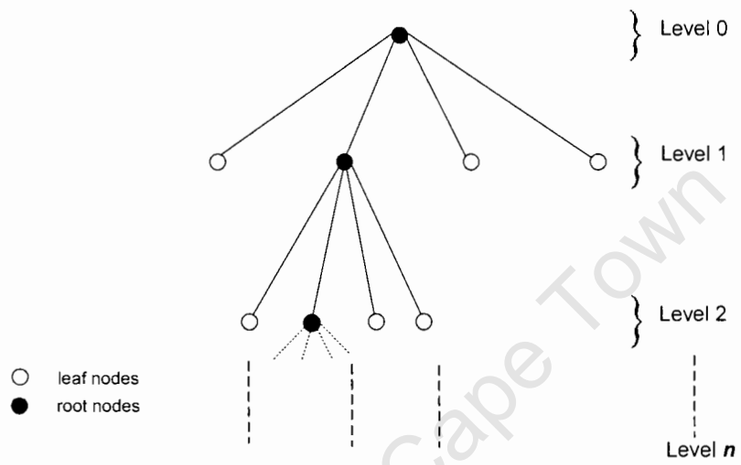


Figure 5.27: Hierarchical modelling

The x, y and z co-ordinate values are not stored directly within the nodes, but are implied by the grid structure. It is necessary though to define the origin of a spatial co-ordinate system that will be linked to the 3-D grid. Gradient and distance field values (for of any other points) can be computed when needed (see section 5.4)

The zero level node contains the following information:

> Node-Zero ($A(x, y, z), \Delta$) where Δ is the grid resolution and A is a grid point with x, y and z co-ordinates.

Frisken et al.(2000) propose the storage of distance field values for all the eight corners of a cell per given node. This approach has the obvious disadvantage of redundancy in data storage values as some grid points are shared by nodes at the same level as well as being shared by the children and parents of the nodes. This research proposes a modified ADF which ensures that data redundancy is eliminated, thus requiring less storage space. The modified ADF approach stores approximately an eighth of the space occupied the general ADF structure (see Figure 5.28 below).

The general data structure of a node is : $nd_i^L((\sum_{j=1}^{1\le j\le 8}\{d_j^m\}), pa, (\sum_{r=1}^{8} k_r))$

Where L = hierarchical level, nd = node, m = object identity, pa = parent node and k = hierarchical node kid.

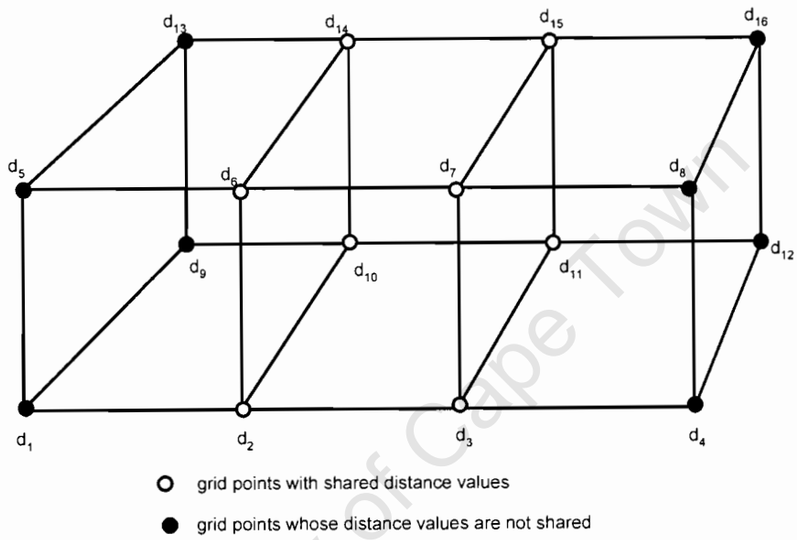


Figure 5.28: Distance field values in a 3-D grid.

Within the general ADFs, the blank nodes (Figure 5.28, above) are stored more than once whereas in the modified ADF, these are stored only once.

This research considers two possible ways in which distance fields are employed in urban modelling:

5.5.1.1 multi-Distance Fields (mDF)

This method can be referred to as the naïve approach. This approach basically provides a distance field map for each and every feature that is in the 3-D scene. Figure 5.29a below,

provides an illustration. The grid point is provided with different attributes as field distances of different features. Distances d_a and d_b are the scalar values from two distance maps, encoded at the same grid point. An adaptive grid is used (see Figure 5.29b) in areas where sharp edges are to be defined (note: the diagram does not provide all areas where the adaptive concept is likely to be applied). The adaptive concept in ***mDF*** approach increases the memory requirements for distance fields immensely. Once an adaptive grid section has been introduced in order to define sharp features of a particular object, all other feature objects (whether distant or close by) will have distances measured to this adaptive grid section as well (plus to each and every grid point). More work is required to improve on the ***mDF*** so that only distances to a particular object are encoded within the object's ADF.

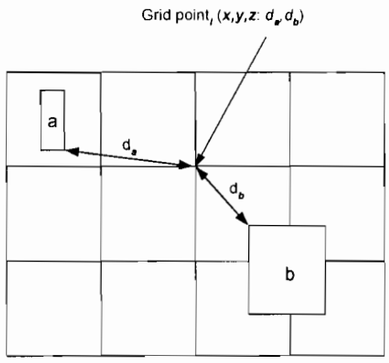


Figure 5.29a. An illustration of two objects being modelled from two different field maps

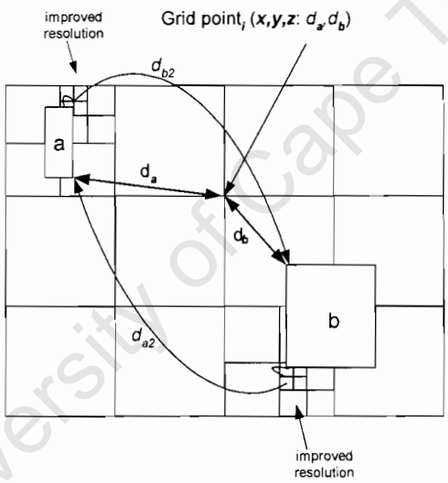


Figure 5.29b. An Illustration of the adaptive grid. Distance d_{a2} to object A is encoded within the ADF of B and similarly, distance d_{b2} to object B is encoded within ADF of A

Space costs: The space cost analysis will be based on a course grid of dimension $r \times r \times r = r^3$ grid cells (note: cells are nodes are the same).

For an r^3 grid, the node count is $(r-1)^3$, assuming no further adaptive cell partitioning takes place, where the node is defined previously in the octree data structure.

Therefore the number of records for the nodes for n objects is $n(r-1)^3$

The distance field values are nr^3.

Assuming m adaptive partitions, each adaptive partition adds 19 grid points and 8 nodes (i.e. these number are obtained form direct counting after each cell is partitioned into eight sub-cells).

Therefore the total number of records is

$$n(r-1)^3 + nr^3 + 27nm$$

Therefore the storage costs scales according to Order $O(n(r-1)^3 + nr^3 + nm)$ (note: the constants have been left out.)

Although the number of urban features is predictably large, the visualisation of the entire city will be implemented using the Level-of Detail (LOD) concept (see Zlatanova 2000) in which limited data sets are rendered for visualization.

5.5.1.2 single Distance Field (sDF)

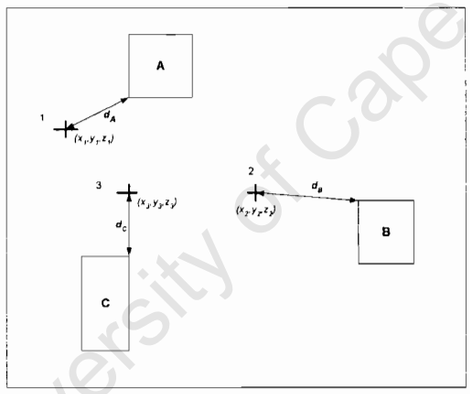


Figure 5.30. Single distance field modelling for urban features.

In single distance field modelling approach, the entire urban scene is represented by a single distance map. Figure 5.30, above illustrates this modelling approach. Distances d_A, d_B and d_C are field values showing the shortest distances to a particular feature. A grid point will only be recorded against a single field value. Other attributes, not shown in the diagram but

essential include the gradient and feature ***ids*** (i.e. ***ids*** of the nearest feature to which the distance field value refers).

Space costs: The **sDF** approach will only have a single field for capturing the distances per grid point. This means that for any number of objects, the amount of space required is a constant. This is equivalent to the space requirements of a single feature in the **mDF** approach.

Therefore space costs for ***sDF*** scales according to order $O((r-1)^3 + r^3 + m)$

5.5.2 Query Operations

A chosen set of queries is investigated for implementation by the DFs. In justifying the use of DFs in the implementation of these queries comparison is made against B-reps, exposing their limitations.

5.5.2.1 Point Membership Classification (PMC).

The most fundamental query in interactive spatial analysis is point membership query, which tests whether a point is contained in a given set or not. In practice, an investigation is made as to whether a point lies on the boundary, outside or inside a closed body:

PMC is defined as $M(p) = \begin{cases} in & if \quad p \in S^o \\ on & if \quad p \in \partial S \\ out & if \quad p \in S^- \end{cases}$

The strings ***in***, ***on*** or ***out*** returned by the function ***M*** tell us whether point ***p*** is inside, on the boundary or outside of S

PMC for B-reps: Conceptually, the simplest algorithm for PMC with respect to a B-rep solid involves casting a ray (i.e., a semi-infinite line) from the point, and counting how many times it intersects the solid's boundary (Samet, 1994). If the number of intersection points is odd, the point is ***in***; if it is even, the point is ***out***. Figure 5.19 shows a 2-D example. Note that the choice of ray used to test a point is arbitrary, and different rays may have different numbers of intersections with the boundary, but for a given point all of these numbers have the same parity, i.e., they are all odd or all even. For example, if we had cast a ray from point **p** to the right in Figure 5.31, the number of intersections would be ***1*** instead of ***5***. Computationally,

instead of a semi-infinite line we use a line segment that is sufficiently long to terminate outside of a box that encloses the solid completely.

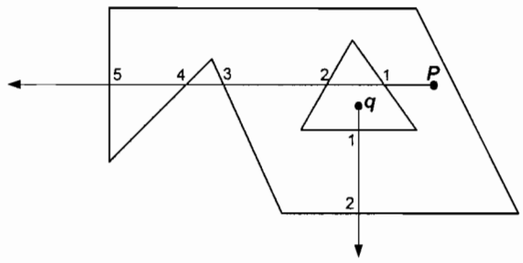


Figure 5.31. PMC for B-reps. The ray cast from *in* point *p* has 5 intersections with polygon's boundary, whereas the ray from *out* point *q* has 2.

Despite its apparent simplicity, there are subtleties associated with this algorithm. First, we also need to consider points that are ***on*** the boundary. How do we count intersections when the endpoint of a ray is ***on***? Secondly, and more perniciously, numerical errors associated with the representation or with the intersection calculations may produce incorrect counts. Third, a ray may be coincident with an edge, vertex or face. How do we count intersections in such singular cases? Figure 5.32, illustrates some of the difficulties. Numerical errors in the endpoint coordinates for the edges of the square are grossly exaggerated in the Figure, to show what can happen in actual computations (note: this scenario is only possible if the B-rep duplicates points). The point **p** is ***in*** the polygon, yet a ray emanating from ***p*** may not intersect any edge (Case 1), or intersect two edges (Case 3), or a vertex (Case 2). In the first two cases the computed number of intersections is even, and the point will be erroneously classified as ***out***. When the ray goes through a vertex, it intersects two edges; should we count one or two intersections (i.e. CASE 3, in Figure 5.32, below)?

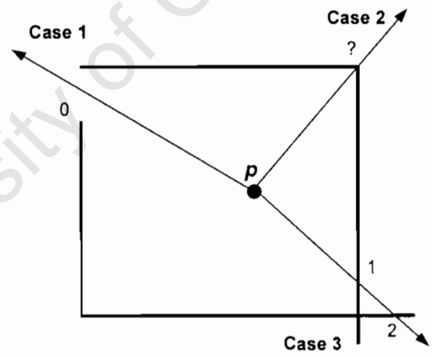


Figure 5.32. Classifying a point with respect to a square whose edge Representations have numerical errors.

The time complexity for PMC(Figure 5.33) is analysed below:

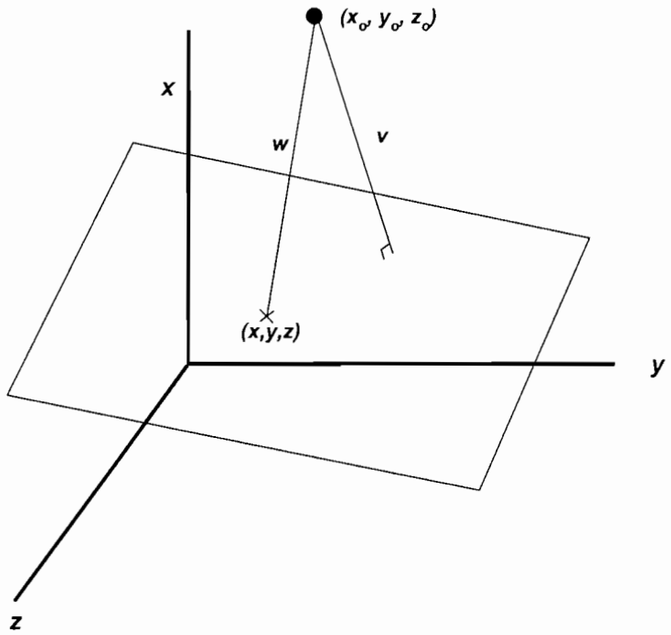


Figure 5.33. Time complexity: PMC for B-reps

The details of the equations are provided in Foley *et al.* (1996). An Algorithm is summarised as:

Read the point $\boldsymbol{Q}(\boldsymbol{x}_o, \boldsymbol{y}_o, \boldsymbol{z}_o)$

Loop 1: from 1 to $\boldsymbol{n}$, where $\boldsymbol{n}$ is the number of planes or faces for the object

Derive the equation of the plane $\boldsymbol{E}_P$

Project point $\boldsymbol{Q}$ *onto Plane* $\boldsymbol{P}_n$

Derive equation $\boldsymbol{w}$ *(see Figure 5.33, above)*

Compute intersection $\boldsymbol{E}_P$ *and* $\boldsymbol{w}$ *within the plane's boundary limits*

Counter $\boldsymbol{m} = \boldsymbol{m} + 1$

End Loop

If $\boldsymbol{m}$ *is odd,* ***Status= the point is inside***

If $\boldsymbol{m}$ *is even,* ***Status= point is outside***

Else, there will be a singularity, which calls for a unique computation (see Laurini,1992)

Return ***Status***

Plane equation parameters derivation for $\boldsymbol{n}$ planes: $\boldsymbol{n} \times \boldsymbol{k}_p$ where $\boldsymbol{k}_p$ is the time for derivation a single plane equation parameters, which is a constant.

Projection of point Q and derivation of parameters for w : $n \times k_w$, where k_w is the time for derivation of equation which is a constant.

Total time for Intersection computation between planes and line: $n \times k_I$, where k_I is the time for intersection determination and it's a constant.

The total time scales as: Order $O(n(k_w + k_p + k_I))$, showing that the time for the computation of the equation parameters, $(k_w + k_p + k_I)$ is crucial to the whole process. $(k_w + k_p + k_I)$, depends on software and hardware used.

Classifying whether a point is lying on the plane (i.e. point on the boundary of an object) is fulfilled by substituting the point co-ordinates into the plane equation. If the conditions of the plane are met, then the point lies on the boundary. The operation scales according to Order $O(k_p)$, which is a constant.

PMC for distance field modelling: General Approach. For point membership classification, we use a function M, which analyses the point in E^3 and gives back the sign of the distance field encoded .

$$M(p_i^l) = \begin{cases} -ve \ then \quad p_i^l \in S^o \\ 0 \ then \quad p_i^l \in \partial S \\ +ve \ then \quad p_i^l \in S^{'} \end{cases}$$

PMC for multiple Distance Fields(*mDF*). PMC for ***mDF*** is a simple operation. For each distance map, the status of the point is tested: Is it inside the feature, on the boundary or outside? A given point will have its distance field value and sign determined through tri-linear interpolation (for off-grid points) before the PMC tests are carried out.

The algorithm for the operation is given below:

Given point p_i *(x,y,z)*
Field Value F= Trilinear interpolation(p_i*)*
If $F = 0$, *then*
Status=on boundary
Else
If $F < 0$
Status=inside boundary
Else If $F > 0$

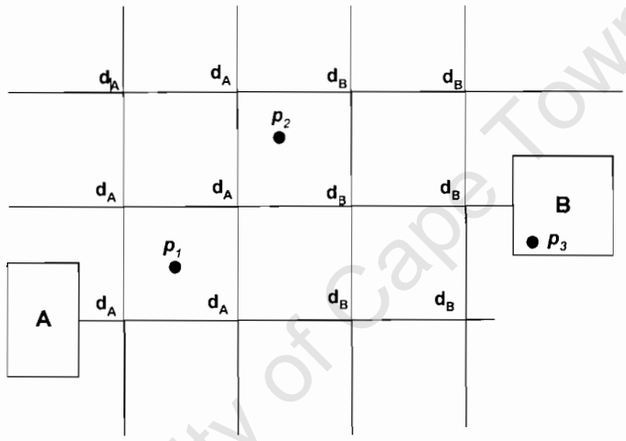
Status=outside boundary

Return ***Status***

Under normal circumstances, the user carries out the above test for a particular point on every distance field map. Assuming that the point is to be tested against every feature and If there are ***n*** features and corresponding maps, then the time complexity for the operations will be *O(**n**)*. Since each object feature has a different field map, a visual check on results of PMC in relation to other objects is made possible by the fact that the maps share the same grid , so objects can be rendered into the same scene (analogous to multiple GIS 2-D layers being overlaid onto each other).

PMC for Single Distance Field (*sDF*). Generally there are three situations encountered during PMC within the ***sDFs*** (see Figure 5.34, below):

- Point p_1 outside object ***A***, but within its distance field
- Point p_2 outside object ***A*** and object ***B***, and lying in a cell within the boundary of the two distance fields (i.e. that of ***A***, and ***B***)
- Point p_3 inside object ***B.***



d_A: distance field value to object A

d_B: distance field value to object B

p_1, p_2 and p_3: user defined points

Figure 5.34. PMC for sDF

Given point p_i *(x,y,z)*
Locate Grid cell for point p_i
If all 8 cell Field values belong to a single feature, Then

Field Value F= Trilinear interpolation(p_i*)*
If F=0, then
Status=on boundary
Else
If $F > 0$

Status=inside boundary
Else If $F < 0$
Status=outside boundary
Else
Point $\mathbf{p}_i$ *is outside all features.*
Return ***Status***

Because the ***sDF*** approach maps all the features within the same map, there is no need to carry out tests in each and every distance map as is the case with the **mDFs** approach. In this instance, the time for the operation for PMC is constant with complexity O(***c***), where ***c*** is the constant overhead. This time is thus not affected by an increase in the number of features being modelled.

5.5.2.2 Line Membership Classification(LMC)

A point ***p*** is either inside, outside, or on the boundary of a reference set ***S*** (i.e. the object). But a candidate set ***X***, that contains a continuum of points may not be entirely contained in either ***S*** or its complement. Therefore, we need a more general notion of membership classification. First let ***L*** be a line segment in $\boldsymbol{E}^3$. We define the membership classification of ***L*** with respect to ***S*** as the function

$$M(L,S) = (LinS, LonS, LoutS)$$
$$LinS = r^1(L \cap S^o)$$
$$LonS = r^1(L \cap \partial S)$$
$$LoutS = r^1(L \cap S^-)$$

Therefore ***M*** divides the line segment into three subsets ***LinS***, ***LonS***, and ***LoutS***, which are, respectively, inside, on the boundary, and outside of ***S***. In the formulas above, $\boldsymbol{r}^1$ denotes regularization in 1-D (Foley *et al.*1996). It implies that the results should be composed of closed line segments with no dangling, or isolated, points.

LMC B-reps: Algorithms for line/solid classification for B-rep objects are very similar to the point classification algorithms. However, the problems encountered in LMC are exacerbated. In PMC, there was the flexibility of choosing an arbitrary line for ray casting, that flexibility is not available in LMC.

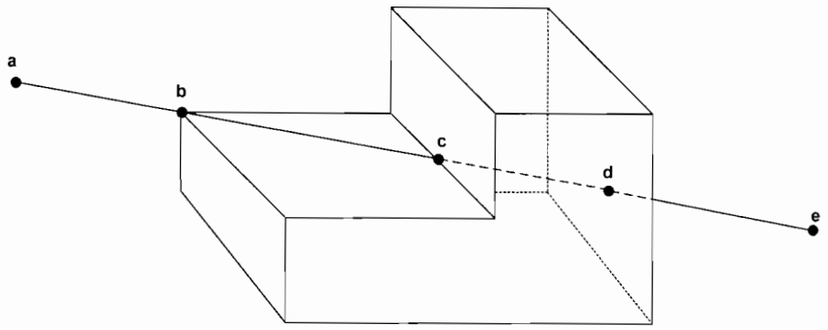


Figure 5.35.LMC: Line through a 3-D object feature

Read end points $Q_a(x_a, y_a, z_a)$ *and* $Q_e(x_e, y_e, z_e)$

Derive the line Equation E_L *for each line segment (Figure 5.35)* ***ab, bc, cd and de***

For each line segment:

Loop 1: from 1 to n, where n is the number of planes or faces for the object

Derive the equation of the plane E_P

Check for intersection between E_P *and* E_L

If number of plane intersections $m = 0$

Status= line segment does not cut through plane

If number of plane intersections $m = 1$

Status=line segment passes through plane

Counter $R = R + 1$

If number of plane intersection $m > 1$

Status=line segment lies on the plane (on object boundary)

End Loop

If R *is odd and greater than 1, Overall Status= one end of the line is inside object*

If R *is even and greater than 0, Overall Status= both ends of the line are outside the object or inside*

If $R = 0$, *Overall Status=object and line segment are disjoint*

Return Overall Status

The computational cost (i.e. for a line segment) for the LMC scales the same as that of the PMC i.e. order $O(n(k_w + k_p + k_l))$.

The major weakness of the above algorithm is its failure to give a single answer to situations when both ends of a line cut through a complicated object (Figure 5.36, below). The algorithm simply suggests that both ends are either outside or inside the object.

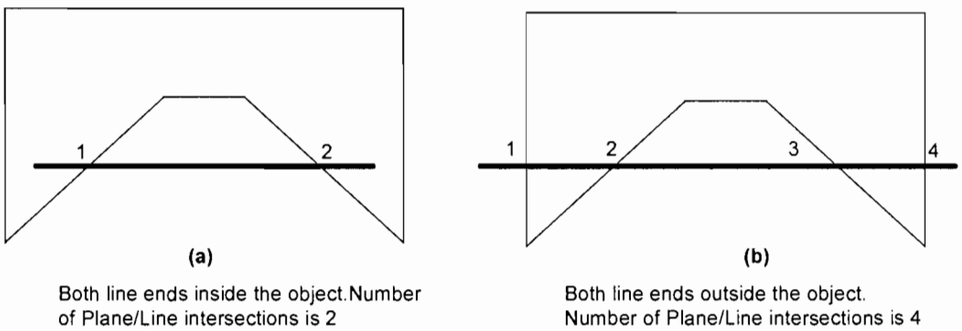


Figure 5.36: LMC: Unique situation: Even number of line/Plane intersections in (a) and (b).

LMC in distance fields. General Approach. As is the case with PMC, a point set that defines the line feature is analysed for its distance field values. Let the line be defined by a point set L, and a point on the line $\boldsymbol{p}_i^l \in \boldsymbol{L}$, where $\boldsymbol{l}$ is the line index.

Figure 5.37 presents the common scenarios encountered during LMC. The **mDFs** and **sDF** are to be tested against these.

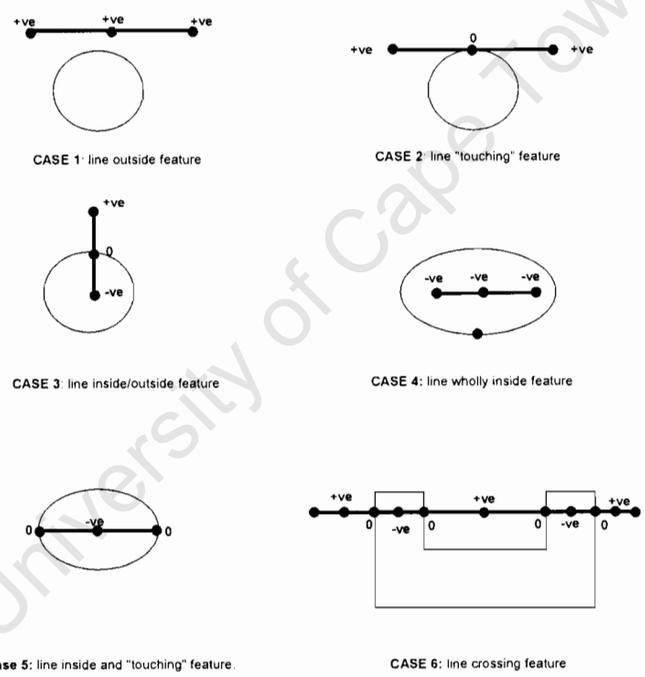


Figure 5.37. Different LMC scenarios

LMC for mDFs: Firstly, the line set members to analyse are determined (Figure 5.38):

In addition to the first (x_s, y_s, z_s) and end point (x_e, y_e, z_e) members of the line, the number of extra members is: $t = \frac{\sqrt{(x_e - x_s)^2 + (y_e - y_s)^2 + (z_e - z_s)^2}}{r_l} - 1$, where r_l is the grid resolution at hierarchical level l .The point co-ordinates of the member elements are obtained from:

$x_i = i\left(\frac{x_e - x_s}{r_l}\right)$, $y_i = i\left(\frac{y_e - y_s}{r_l}\right)$ and $z_i = i\left(\frac{z_e - z_s}{r_l}\right)$, where i is the point member number.

The analysis algorithm:

Read in the end point co-ordinates of the line
Determine the intermediate points
For each point, find the node id

If node has kids then

Add more members points either side of the point by $r_l/2$

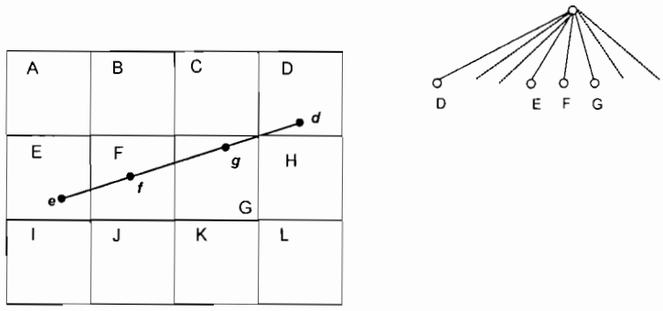
For each point, find the distance value dj *through tri-linear interpolation*
Investigate the sign of line set members
if $\mathbf{M}(\partial\bar{L}) > 0$ *,where* $\mathbf{M}$ *is the sign classification function.*
Then status = Line is wholly outside the object
If $\mathbf{M}(\partial\bar{L}) < 0$
Then Status= Line is wholly inside the object

If $\mathbf{M}(\partial L) < 0$, $\mathbf{M}(L^o) > 0$ *and* $\mathbf{M}(\partial L) < 0$
Then Status, Line cuts through feature with end points in the object
If $\mathbf{M}(\partial L) > 0$, $\mathbf{M}(L^o) < 0$ *and* $\mathbf{M}(\partial L) > 0$
Then Status=Line cuts through the object with the end points outside.

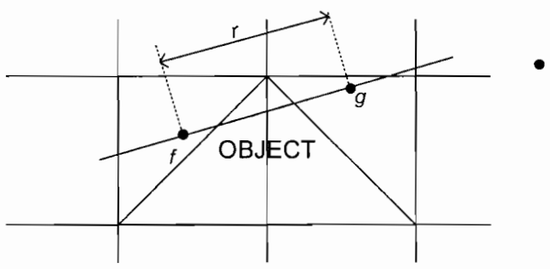
Return Status

It can be seen that different modelling scenarios can be defined and captured in the ***status*** variable.

Figure 5.38(b) shows the special case. If the line point sampling is maintained at r resolution, then the status of the line segment between points f and g (in figure 5.38(b)) will show that the line is outside the object. The remedy is to increase the cell resolution to $r/2$ in between line segments encountering this special case.



(a) Normal case



(b) Special case: object "cutting through" line between ***f*** and ***g***

Figure 5.38: LMC-mDF/sDF

LMC for *sDF*: The line member determination approach for ***sDF*** (Figure 5.39) is similar to the ***mDF*** approach above. The algorithm however includes the analysis of distance field object attributes (i.e. investigates the object tags assigned to field values).

The analysis algorithm:

Read in the end point co-ordinates of the line
Determine the intermediate points
For each point, find the node id

If node has kids then

Add more members points either side of the point by $r_l/2$

Investigate the distance value tags around the cell (8 values)
If tags heterogeneous then
Status = line segment is outside features
If tags are the same, then
For each point, find the distance value dj *through tri-linear interpolation*
Investigate the sign of line set members
if $M(\partial\bar{L}) > 0$ *,where* ***M*** *is the sign classification function.*
Then status = Line is wholly outside the object
If $M(\partial\bar{L}) < 0$

Then Status= Line is wholly inside the object

If $M(\partial L) < 0$, $M(L^o) > 0$ *and* $M(\partial L) < 0$
Then Status, Line cuts through feature with end points in the object
If $M(\partial L) > 0$, $M(L^o) < 0$ *and* $M(\partial L) > 0$
Then Status=Line cuts through the object with the end points outside.

Return ***Status***
Else

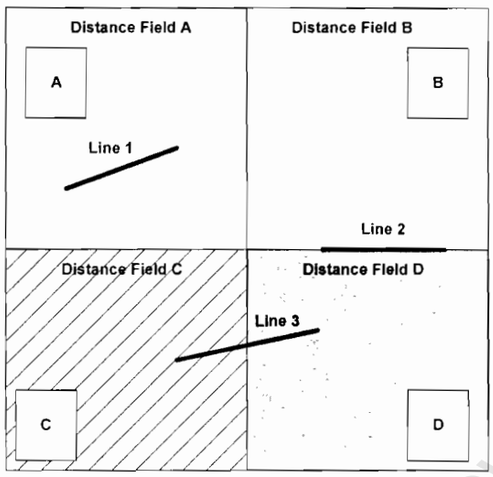


Figure 5.39.Line membership classification for sDF

5.5.2.3 Location distances

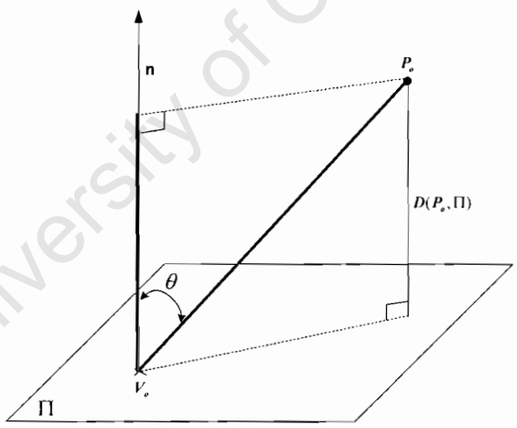


Figure 5.40: Point to plane distance

The location distance (Figure 5.40) in this research refers to the distance from any point of interest to the nearest point on a specific object. This distance is readily provided in a distance field map which only represents a single feature (see ***sFD***, below), but the situation is different in ***mDFs***. In B-reps, the approach would involve a number of mathematical computations which would make interactive querying unattractive.

The distance from an arbitrary 3-D position $\boldsymbol{P_o(x_o, y_o, z_o)}$ to a position $\boldsymbol{V_o(x_v, y_v, z_v)}$ on the plane $\boldsymbol{\Pi : (ax + by + cz + d)}$ (see diagram 5.41, below), can be computed from the ***dot product***:

$$\boldsymbol{D(P_o, \Pi) = |P_o - V_o| \cos\theta = \frac{n.(P_o - V_o)}{|n|} = \frac{ax_o + by_o + cz_o + d}{\sqrt{a^2 + b^2 + c^2}}}$$ where $\boldsymbol{n}$ is the normal vector. $\boldsymbol{|P_o - V_o|}$ can be determined.

The algorithm can be summarised as:

Read point co-ordinates $\boldsymbol{P_o(x_o, y_o, z_o)}$ *(Time* $\boldsymbol{k}$*)*

Read point co-ordinates $\boldsymbol{V_o(x_v, y_v, z_v)}$ *(Time* $\boldsymbol{k}$*)*

Generate Equation of Plane $\boldsymbol{\Pi : (ax + by + cz + d)}$ *(time* $\boldsymbol{E_p}$*)*

Determine $\boldsymbol{D(Po, \Pi)}$ *(Time* $\boldsymbol{E_D}$*)*

Compute $\boldsymbol{|P_o - V_o|}$ *(Time* $\boldsymbol{E_{P_o - V_o}}$*)*

The algorithm is made up of relatively complex mathematical operations:

The time is a constant for $\boldsymbol{n}$ number of planes is given by $\boldsymbol{O}\, n(2k + E_p + E_D + E_{P_o - V_o})$

Point Location: *mDF.* Point location in mDFs is a simple issue of computing the distance field value of a point located within the distance map through tri-linear interpolation. One has to note that the computation is carried out within the "right" distance map of the feature which is being queried.

Point Location: *sDF:*The point location based queries pose some challenges in the sDF scheme. Contrary to the approach of mDFs, above , in this instance, a point of interest could be located within the field of a feature *A*, and queries made to a distant feature *B* (see Figure 5.41, below).

The most important questions to be asked are:

- How do we obtain the distance $\boldsymbol{d_e}$?

- How do we get to know that the point q is within region A (not B) and thus cannot calculate the location distance d_a from simple tri-linear interpolation ?

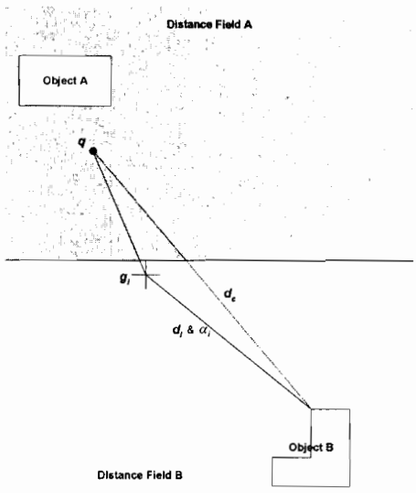


Figure 5.41. Location distances computation

A nearest search algorithm (i.e. search for a point in Distance field B, closest to q) is carried out and a point g_i located (see Appendix B: algorithm B1). Once g_i has been obtained, the distance (d_i) to feature B can be read from the distance map and the direction (α_i) computed from the central differences approach (see Appendix C). A co-ordinate function is then applied to compute the co-ordinates of the nearest feature point to g_i, on the surface object B i.e.

$C(x_g, y_g, z_g, d_i, \alpha_i) = (x_b, y_b, z_b)$, where α_i
is the derived direction to nearest point feature on B

And the distance d_c is computed from:

$$d_c = \sqrt{((x_g - x_b)^2 + (y_g - y_b)^2 + (z_g - z_b)^2}$$

The nearest neighbour point g_i is obtained from a *nearest neighbour search* process (see Appendix B: Algorithm B1)

The time complexity for the operation is made up of the search operation for point g_i and determination of distance (d_c). The search operation scales as $O(dn(\log n))$ where n is the number of points in the search space and d is the octree hierarchical level (Arya *et al.*, 1999). The total time complexity is $O(dn(\log n)+k)$ where k is constant time for distance (d_c) computation.

5.5.2.4 Buffering

The buffer operation for 3D geographic features plays an important role in decision making processes. 3D buffering on point-shaped features can be applied for the analysis of 3-dimensional pollution areas caused by point sources. 3D buffering on line-shaped feature can be used to decide the position for laying water-pipes, gas-pipes, electric cables, or telephone cables underground. Buffering on 3D body features can be utilized for city-planning and landscape architecture simulation. Buffer design operations for H3-D are discussed under different feature type categories, below:

Buffering of Point Data: Buffering point data is the simplest form of buffering (in B-reps, 2.5D representations, etc), as the process simply involves the creation of a sphere about each point of radius equal to the buffer width. In distance field modelling (**sDF** and **mDFs**), the process involves the offsetting of distances from the point feature by an amount equal to the buffer distance and a surface is then developed (see Figure 5.42 , below). The surface shape is affected by the resolution of the distance field grid and as a result, one would get a more realistic buffer shape through the 2.5-D and B-rep approaches.

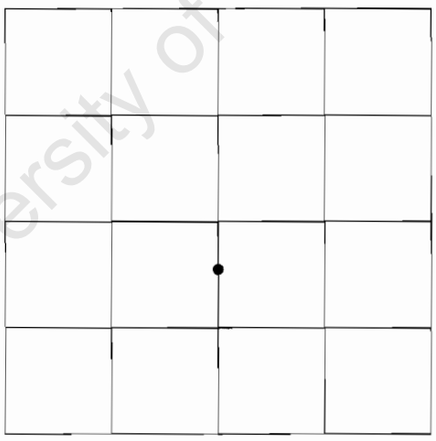


Figure 5.24. Buffering a point in a distance field.

Buffering of Line Data: The algorithm for buffering line data (for B-reps) is more complex than that for buffering point data, as lines can be made up of multiple segments (Worboys,1994). Through a series of line equation calculations and intersection tests, a 3-D volume can be generated around a line as a buffer. For distance field modelling, the same principle of distance offsetting is applied and a surface generated (see Figure 5.43, below). The marching cubes algorithm is applied (see appendix C), to generate the surface of the body object around the line.

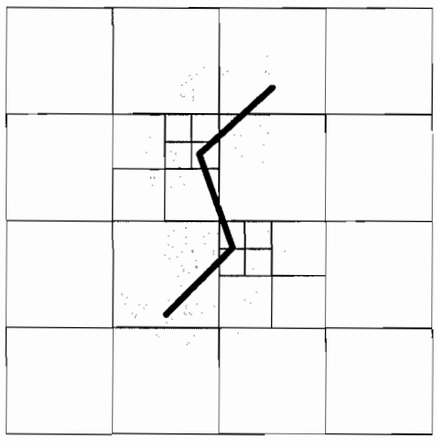


Figure 5.43. Buffer of a line feature

For B-reps, buffering of linear features is computed for this basic approach:

Assume a buffer width, b (see Figure5.44, below).

The end points of parallel buffer lines are thus:

$$x_a = x_1 - b.\sin\left[\tan^{-1}\left(\frac{\Delta x}{\Delta y}\right)\right]$$

$$y_a = y_1 + b.\cos\left[\tan^{-1}\left(\frac{\Delta x}{\Delta y}\right)\right]$$

$$z_a = z_1 + b.\sin\left[\tan^{-1}\left(\frac{\Delta z}{\Delta y}\right)\right]$$

and

$$x_b = x_1 + b.\sin\left[\tan^{-1}\left(\frac{\Delta x}{\Delta y}\right)\right]$$

$$y_b = y_1 - b.\cos\left[\tan^{-1}\left(\frac{\Delta x}{\Delta y}\right)\right]$$

$$z_b = z_1 - b.\sin\left[\tan^{-1}\left(\frac{\Delta x}{\Delta y}\right)\right]$$

The co-ordinates values obtained will provide the outline for the construction of a cylinder around the line segments. Where two line segments meet (see Figure 5.44b), gaps and overlaps are created within the buffer cylinder. This makes buffering of lines within the B-reps unattractive. This is further compounded by the number of mathematical equations generated at each point (i.e. six.). For a line with n nodes, the computation of the buffer will scale according to Order $\boldsymbol{O}(6(\boldsymbol{n}+\boldsymbol{k}))$ where k is the constant read time for values at each point.

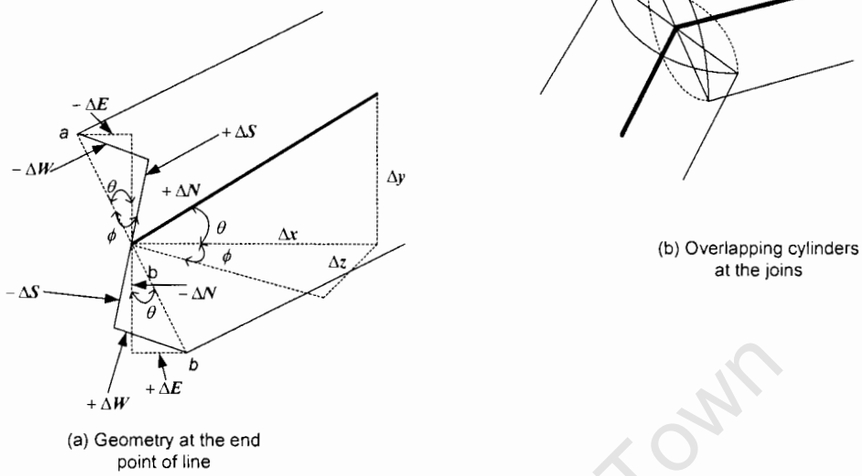


Figure 5.44: Line Buffering

Buffering of Body features: Body feature buffering remains a topical issue in 3D GIS. Attempts made so far have been to buffer models formed from primitive instances. In this particular case, the models will be quite regular (though not fully representative of the urban feature), and a buffer is simply formed by changing the object parameters (see Figure 5.45, below)

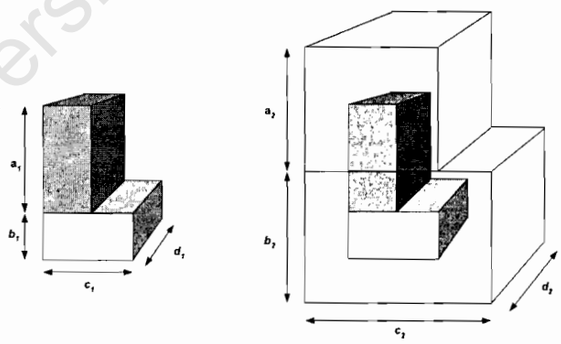


Figure 5.45. body feature buffering-2.5D

Attempts for 3-D buffering of lines are a complicated issue in B-reps, as highlighted above, so one would expect body feature buffering to be even more complicated. This research makes maximum use of the offsetting quality of distance fields in implementing 3-D buffer (see Figure 5.46,below).

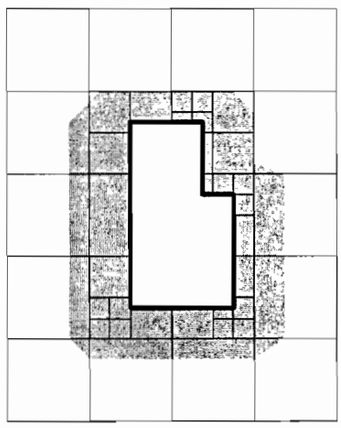


Figure 5.46. Body feature buffering in mDF

Distance buffering in ***mDFs*** creates a buffer field around each object, and in separate scenes. Relative analysis operation between objects cannot be carried out but buffer zones are of unlimited radius.

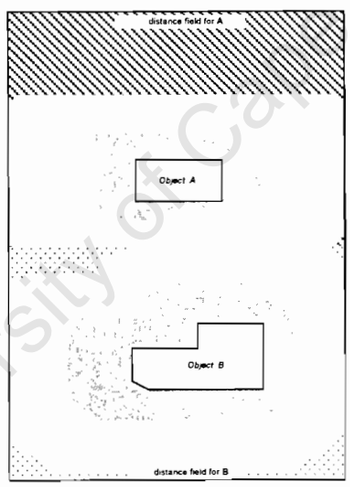


Figure 5.47. *sDF* Buffering of entire scene without field interpolation

Buffering in ***sDF*** has a resemblance to that of 2-D systems. There are two possible instances in which the buffer operation is implemented. Firstly, the buffer can be generated around all

the objects scene simultaneously (see Figure 5.47, above), this would simulate a query of the form "create a buffer 2meters around all features". Buffer zones of particular objects are limited to the respective distance field areas. The second approach generates buffers which overlap into neighbouring scenes. This is only possible through the extrapolation of distance fields onto the grid points of the neighbouring fields(Figure 5.48,below). For interpolation, the grid scalar values are computed using a similar approach to section (5.3.2).

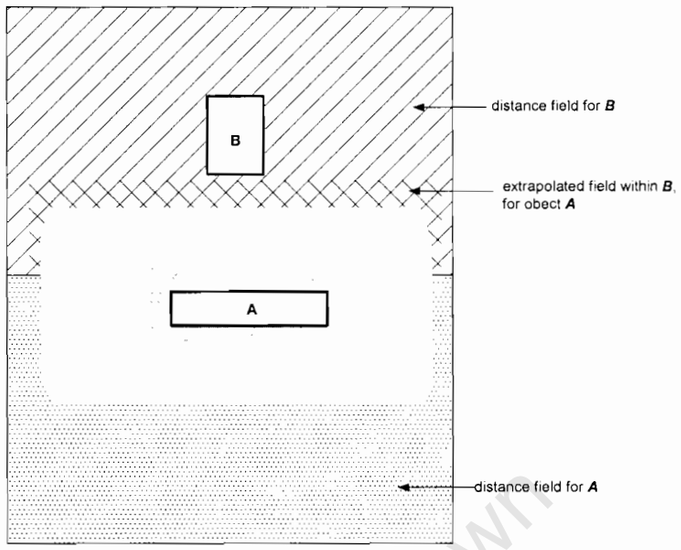


Figure 5.48: sDF with Field extrapolation

5.5.3 Justification for using sDF

A strikingly unattractive feature of the ***mDFs*** approach is the need to "toggle" between different feature distance maps when making analysis. This is analogous to having a theme layer for each and every feature in a 2-Dimensional GIS. The number of layers, and hence the cost of spatial analysis operations will grow with every feature mapped. An analysis based on at least two feature elements would mean that the two distance field maps will need to be merged. The space costs are enormous with the ***mDFs*** approach since for every feature mapping, the whole grid elements will need to encode scalar values radiating from it. On the other hand, ***sDF*** has a single field for all urban features. The analysis will be carried out on this single field and bi-analysis operations are easily implemented. A line feature "cutting" through two or more feature objects would pose some challenges in ***mDFs*** as there is a need to analyse multiple fields. Buffer analysis is useful if implemented in the vicinity of other object features, so with ***mDFs***, this would be complicated to achieve. Although the extrapolation of distance field has been shown as necessary in instances where the buffer

extends into another object, this is hardly applied in analysis as the most common approach involves the simultaneous generation of buffer zones.

As a result of the weaknesses exhibited by the ***mDFs***, this research has formulated the hybrid representation between the B-rep and the ***sDF***. It has to be noted that the ***sDF*** is not a normal DF as it includes feature attributes and as a result, it has a set of new algorithms (note that this is something new in DFs).

Chapter 6
Prototype: Interactive 3-D modelling system

This chapter focuses on the implementation of the concepts discussed in the previous chapters, i.e. interactive data retrieval and visualization over the web presented in chapter 4, the 3-D data representation proposed in Chapter 5 and some of the spatial analysis operations outlined in the same chapter. The chapter will "see" a *prototype* system developed.

Hawryszkiewycz (1994) defines a prototype as "an early model of a product or part of, which is tested so that the design can be changed if necessary".

The main components of the proposed system architecture (see chapter 4) are an application web server, database server and the client. The components were chosen in accordance with the fundamental consideration for a low cost and easy to implement solution. In general, "proof-of-concept system architecture requires minimal investments" (Zlatanova, 2000). Therefore the selection of components was limited to software and hardware currently available at the University of Cape Town and off-the- web plug-ins.Throughout this chapter, motivation for the use of different softwares will be provided.

6.1 The prototype system: components

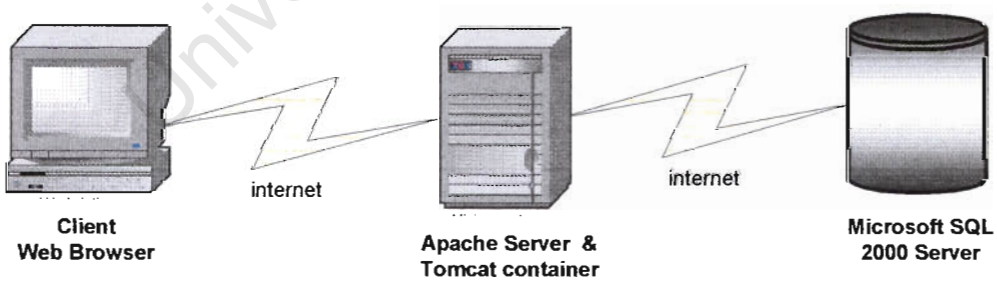


Figure 6.1. Three tie-architecture showing the link between the main components of the web-based application.

The client/server architecture of the prototype is designed in the light of the integrated distributed approach. It has a typical three-tier configuration (Figure 6.1, above).

A client composed of VRML browser and EAI residing in the Web browser: Internet Explorer or Netscape Navigator. The client provides several WebPages embedded with Java applets, which allow users to select operations, define properties of 3-D scenes or input parameters for 3-D analysis and make queries to a remotely located database. The middle-tier is an application server (Apache server integrated with *Tomcat container*) and finally a Database server (Microsoft SQL2000 Server).

A detailed analysis of the three tiers is provided below:

6.1.1 SQL database server

The Database server provided for this research is the Microsoft SQL 2000. The University of Cape Town enjoys a site license for its use. It's located within the Computer science department. Microsoft SQL Server 2000 is a relational database management system (RDBMS), which serves to provide data storage and management services, along with data access support for Internet-based users. SQL Server 2000 accepts and executes client requests for the addition, modification or deletion of data, as well as commands for creating objects such as databases and tables (i.e. using the UPDATE, SELECT and DELETE SQL statements). The Server allows the user to retrieve and arrange data in a relational way, efficiently storing data in the form of rows and columns.

A significant enhancement is the SQL Server's use of XML format files for Internet use. XML is an Extensible Mark-up Language that is designed to improve the functionality of the Web by providing more flexible and adaptable information identification. Because of XML, different "mark-up" documents can be entertained e.g. PHP, HTML, JSPS and others) In earlier versions, Web programmers had problems in designing Web pages that displayed the stored data. Users can specify the SQL syntax in the browser and the queried data is returned by the SQL Server as an XML document. The requested data is returned to the Web in the XML format. SQL Server 2000 can be installed on a computer with any Intel-based or Alpha-based processor. It can be installed on Windows 2000 or Windows NT operating system Since SQL Server 2000 is closely integrated with the Windows 2000 operating system, it can take full advantage of Windows 2000's many performance-enhancing features such as using multi-threading to support clients. This means that a thread is assigned for each client, significantly enhancing the ability of the SQL Server to support a large number of clients. SQL Server 2000 has a family of components, which

work together, to allow distributed implementation. One of these components is the SQL Enterprise Manager (SEM). This tool provides a GUI interface, which enables a Database Administrator to administer instances of SQL Server running on a local computer or at a remote location. Other important components of the SQL Server 2000 family include SQL Compiler, the database engine and the Network Access Managers. SQL Compiler, which is an integral part of the SQL Server architecture, accepts SQL statements as input and compiles them. The database engine accepts and stores data in tables, accepts queries, processes them, and returns the required data to the user. Queries can be written in Structured Query Language

Microsoft SQL Server 2000 can implement storage in a two-tier or multi-tier (n-tier) environment. In a two-tier environment, client applications access data on the SQL Server over the network, with the business rules component present on the server. In a multi-tier environment, the client application accesses the SQL Server over the network, but the business rules are present on another server. SQL Server 2000 also provides support for the Web. With the help of Internet Information Server (IIS) service, data requests from the Internet can be handled with ease. The most important benefit of SQL Server 2000 as a client/server system is its capability to cater to a huge number of clients simultaneously. In this research, database connection to the server is implemented through the JDBC-ODBC bridge driver.

6.1.2 Application Server

The second tier, the application server, is made up of the *Apache Server* integrated with the *Tomcat container*. Resident on this application server are VRML and VTK generated files formats for the 3-D model (see section 6.2).

Central to the operations of the application server are servlets. The servlet acts as the middleware to query and update the database, and it also returns data to the browser as HTML. The applications server accepts requests from browsers like Netscape and Internet Explorer and then returns the appropriate HTML documents.

In order to run servlets, the web server must be servlet enabled. For this research, we employed the Apache web server and the Tomcat servlet engine version 4.1.24.

6.1.2.1 Apache Server

Apache has been the most popular web server on the Internet since April of 1996. The July 2003 Netcraft Web Server Survey found that 63% of the web sites on the Internet are using Apache, thus making it more widely used than all other web servers combined (Holzner, 2002). The name "Apache" appeared during the early development of the software because it was "a patchy" server, made out of patches for the freely available source code of the NCSA HTTPd Web server. The Apache server is freely available and is based on the HTTP protocol. Apache is known to compile on just about every UNIX variant: Solaris 2.X, SunOS 4.1.X, Irix 5.X and 6.X, Linux. It can do the same on Windows NT. The requirements for available hard disk space (1.5 MB), processor (at least Pentium 486DX) and RAM (at least 16MB) are moderate (Holzner, 2002).

Integrated with Apache server, is the Tomcat container.

6.1.2.2 Tomcat container

Tomcat is Web Container (Figure 6.2 below), something that acts as an executable environment for Java Server Pages (JSP). Java Server Pages are executable Java program elements that are embedded into HTML pages. These are stripped out and compiled up on the fly as Servlets, Java programs linked to HTML pages, allowing dynamic content in HTML pages. Tomcat is a "pure"-Java example of such a container. It is, with the exception of one or two scripts, entirely written in Java. As such a binary distribution for version 4.1.24 was downloaded and installed. The configuration issues to allow connection between it and Apache were integrated in the installation.

Figure 6.2. Apache-Tomcat server starting-up

Tomcat can easily hook into Apache, Microsoft IIS, and Netscape Enterprise Web servers, as the reference platform for current servlet and JavaServer Pages (JSP) standards.

Tomcat has no built-in development tools: Programmers can use their favourite Integrated Development Environment (IDE) or rely on the command line Java compiler from Sun.

The main benefit of using the Apache server and Tomcat servlet engine is that both of these products are open source. These products can be downloaded and installed for free, they run on a variety of computer server hardware platforms which are scalable and supported and enhanced by developers throughout the world.

6.1.3. The Client

The client is made up of two components: The topological, which is the main component and the Distance Field component (see section 6.3.4, below). . The main component interface is through Microsoft Internet explorer or Netscape Internet browser. In the web browser, a JSP form is used to get user-input and display 3-D models and textual query results of the database.

Embedded in the Client browser are: VRML browser or plug in and a Java applet. There are basically two VRML browsers that were discovered during the course of this research as being Capable of EIA implementation, The Blaxxun and Cortona Browsers. The Blaxxun virtual environment client, CCPRO, is a free Web browser plug-in which can be downloaded from the Blaxxun Web site, however, it is only available for Windows 95/NT platform and applicable to Netscape internet browser. This limits its usefulness in a heterogeneous distributed environment envisaged by this research work. Cortona VRML plug-in was the preferred choice to allow for 3-D interactive visualization. The plug-in provides more efficient VRML visualization and memory usage. The plug-in works in both Netscape and Internet explorer browsers. It comprises three elements, the three-dimensional view, a chat window and a control panel. The three-dimensional aspect of the system utilises VRML versions 1.0 and 2.0.

The applet provides an interface between the user and the 3-D scene (in Cortona) through the EAI.

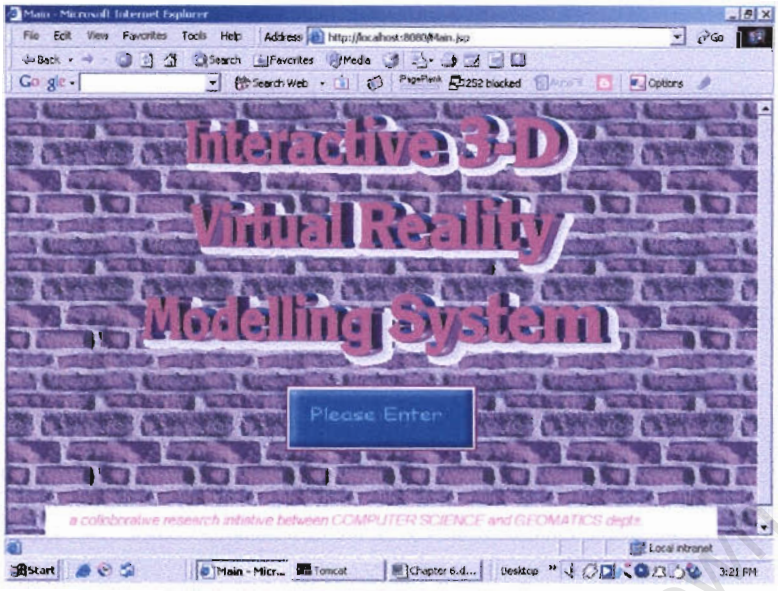


Figure 6.3. Application's Interface Page

The client GUI is one of the key fundamentals of the whole project (see Figure 6.3,below); it enables the user to control the major operational modes, make specific input selections, and to navigate within the application. The GUI is composed of a main menu page and a number of sub-menus.

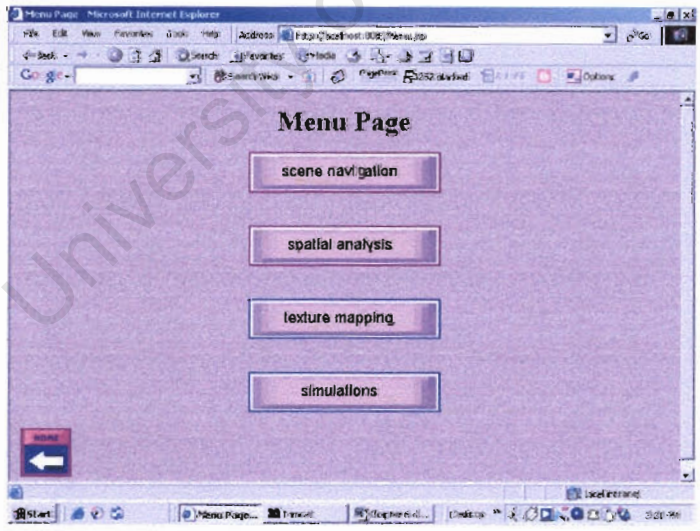


Figure 6.4(a). The main menu architecture

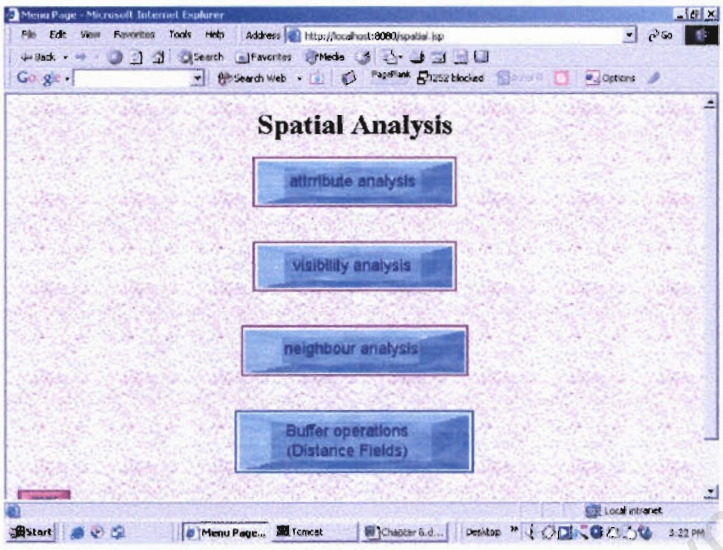


Figure 6.4(b). Interface showing the sub-menu options

Figure 6.4(a) and Figure 6.5(b) give the architecture of the main and sub-menus for the application, respectively. When a particular option or function is selected in the menu, an encoded message is communicated to relevant modules in the visualization. The fundamental building blocks of the menu are two buttons that enables textual or object search.

6.2 3-D model generation

The generation of a high quality 3-D model for a real world object includes several, partially independent steps. Figure 6.5 below, shows an overview over these steps. The object geometry data are obtained in "spaghetti" format from CAD applications and are captured together with attribute data, manually into the Database tables. The object texture files are referenced to their locations in the relational tables. Depending on the nature of spatial analysis operations to be performed, the objects data is either passed from the database onto the VRML generator or VTK generator within the application server. The VRML based model is displayed to the user through the VRML plug-in in the browser. As for the VTK based model, a separate interface for the prototype is invoked from the Visualisation toolkit.

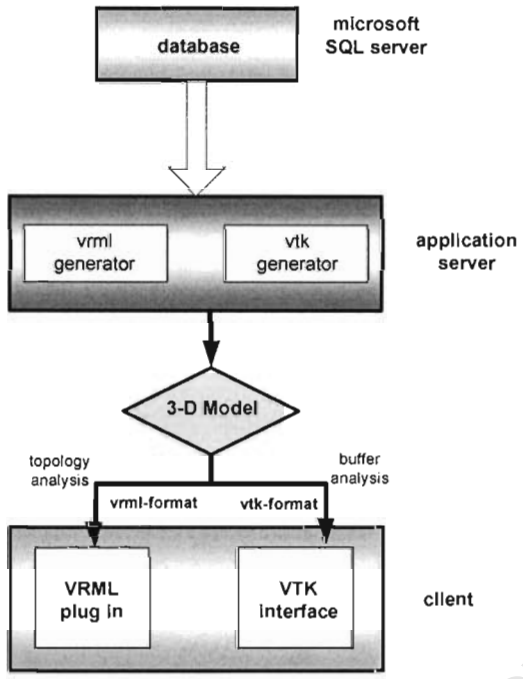


Figure 6.5. Showing the links between the database and VRML/VTK generators and the client

6.2.1 VRML format

VRML is basically a language to describe 3-D models and to make them accessible on the Internet. It is an open standard and can be used without licensing fees. Visualization/interaction is done by plug-ins for web browsers (Netscape Communicator, Microsoft Internet Explorer). The first version of VRML was developed in 1994 and was nearly identical with the Open Inventor ASCII file format. VRML 1 was sufficient to describe static 3-D models and to link their objects with other VRML or HTML documents. VRML 2 was accepted as an ISO standard in late 1997 (VRML'97) and has been called VRML'97 since then. Basic use of VRML is essentially that of dynamic visualizer of pre-formatted files in the browser, without interacting with geo-objects. However, the aim is not only to get a personal view of the 3-D model data but also to interact with the geo-objects.

In The prototype, we use the EAI method for extending interaction with the VRML scene generated by the 3-D visualization or analysis module.

VRML worlds are primarily defined by nodes that describe shapes, sensors and scripts, linked by routes, which pass messages between nodes. Most nodes are of predefined structure containing fields and events that describe their appearance and behaviour. The VRML Script node provides a

general-purpose node for programming new sensors and interpolators for VRML whereby appearance and behaviour of objects in the scene may be modified and defined. Script nodes contain references to the appropriate Java. Class file to call when the script is initialised.

Shape geometries can be basic (boxes, cylinders and cones) or general. The nodes for shape description which were of paramount importance in this research were those for general shapes: the IndexedFaceSet, IndexedLineSet and IndexedPointSet. The IndexedFaceSet defines the form of an the object as a collection of simpler polygons, IndexedLineSet defines the feature as a collection of simpler arcs or edges and the IndexedPointSet defines as a collection of vertices (in X,Y,Z, Euclidean space).

6.2.2 VTK format

Visualisation Toolkit is an open source application. The choice of VTK comes primarily from the fact that it is freely available off the web and there is a consortium waking on nightly updates to the open source. Ideally, the prototype should have integrated VTK within the web-browsers but owing to the fact that the application has only recently been developed for Java and the distributed computing environment, so by the time of putting this research work together, it was not yet fully operational. As such, the interface for 3-D buffer analysis is installed on the client side as a VTK application, but it "feeds" off from the application server.

VTK open-source software was developed by Kitware Inc. It is large and complex tool for data visualization. In version 4.2, it contains more than 800 classes for various purposes. The concept of VTK makes the library easy to use once a programmer has mastered its object-oriented design principles. A large number of developers participate on the bug fixes and/or developing of new useful modules, tools and features. The current version of the VTK can be installed on MS Windows and almost all UNIX-based systems. Therefore there is a good portability on source code level. The Visualization Toolkit is aimed at scientific data visualization and image processing. It contains wide variety of Filters (algorithms), Importers, Exporters, and Renderers and also a set of classes for data representation. The Visualization Toolkit consists of two basic subsystems: a compiled core and a wrapper layer (See Figure 6.6). The core layer is a compiled *C++* class library. In case of MS Windows it is distributed as a set of dynamic linked libraries (.DLL), which are necessary for running any application that uses VTK. The official wrapper layers make it possible to use VTK in the Java (under development), TCL and Python

programming languages. Detailed description of the VTK application is well covered by Schroeder (2001).

Of prime importance to this research work are the Manhattan and Matching-cubes filters (see Chapter 4 and Chapter 5). The Manhattan filter is part of the VTK generator (Figure 6.5 above). It creates a distance map from the Object description data of the Database. The Matching cubes algorithm then creates the model iso-surface from the generated distance map.

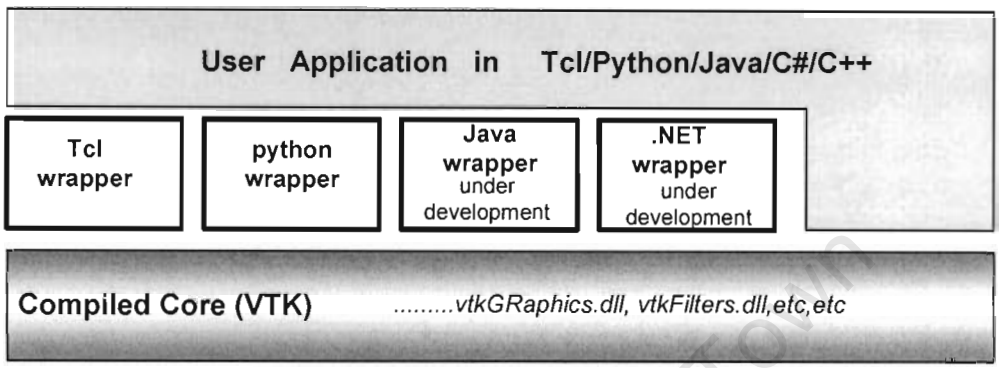


Figure 6.6. showing the VTK architecture

6.2.3 Anatomy of a request-response operation

The process user interacting with the web application can be described from two perspectives: where the user needs textual data without 3-D model visualisation and where the user needs both. In the case for both, the user interacts with the 3-D model to get objects ***ids***, to make spatial analysis. And when the user request information or makes queries on the remote database, a JSP page is executed by a JSP engine, which is installed in a web server or a JSP-enabled application server. The JSP engine receives requests from a client to a JSP page, and generates responses from the JSP page to the client. JSP pages are typically compiled into Java Servlets. The page developer has access to the complete Java application environment, with all of the scalability and portability of the Java technology-enabled family. When a JSP page is first called, if it does not yet exist, it is compiled into a Java Servlet class and stored in the server memory. This enables very fast responses for subsequent calls to that page. (This avoids the CGI-bin problem of spawning a new process for each HTTP request, or the runtime parsing required by server-side includes.) .In this implementation, the servlet invokes the JDBC-ODBC Bridge to get information from a database.

Figure 6.7 illustrates the interaction between these processes. Note that the server and client machines are not necessarily on different machines. For example, a stand alone system can be configured to run Apache while at the same time the client browser can request documents from the server.

summary of operations:

1. The user goes to a web site made using JSP. The user goes to a JSP page (ending with .jsp). If it is spatial analysis and visualisation, the user will interact with the 3-D model, as for database requests, the web browser makes the request via the Internet.
2. The JSP request gets sent to the Web Application server.
3. The Web server recognises that the file required is special (.jsp), therefore passes the JSP file to the JSP Servlet Engine.
4. The next step is to generate a special Servlet from the JSP file.
5. The servlet sends a request to the database server and receives a response.
6. HTML from the Servlet output is sent via the Internet to the client.
7. HTML results are displayed on the user's web browser.

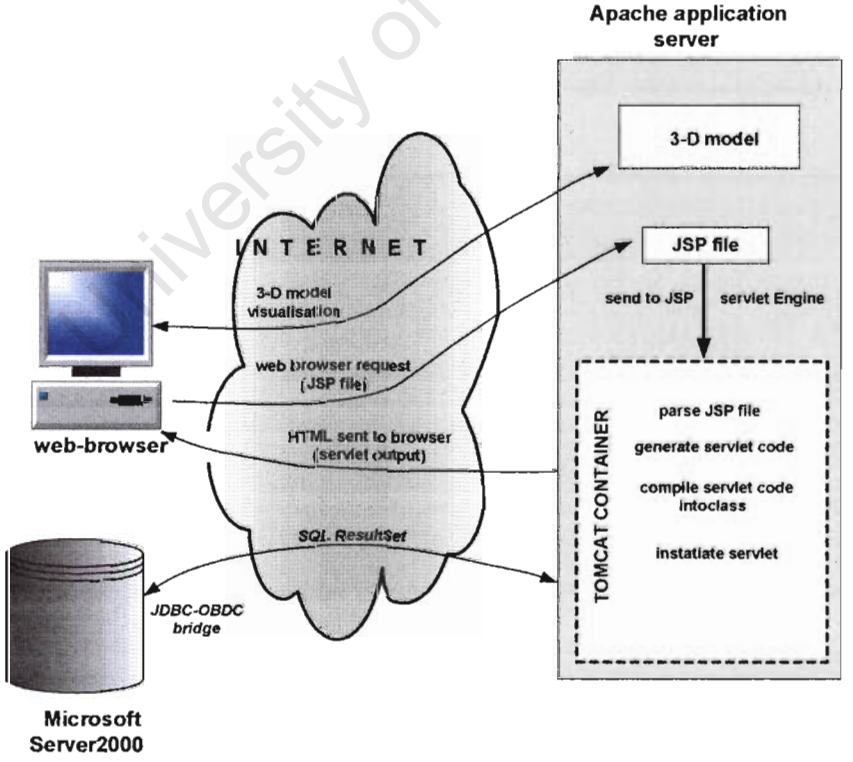


Figure 6.7. Summarise the anatomy of a request-respond process

6.3 Spatial analysis

Although in the conceptual design, a number of spatial analysis operations are highlighted, in this prototype, only some of these are implemented as proof-of-concept.

6.3.1 Attribute analysis and 3-D model visualisation.

One of the most important qualities of geo-information systems is the ability to assign attribute information to spatial objects. In this web-based application, the user is presented with a GUI (Figure 6.8, below) with a number of options on it:

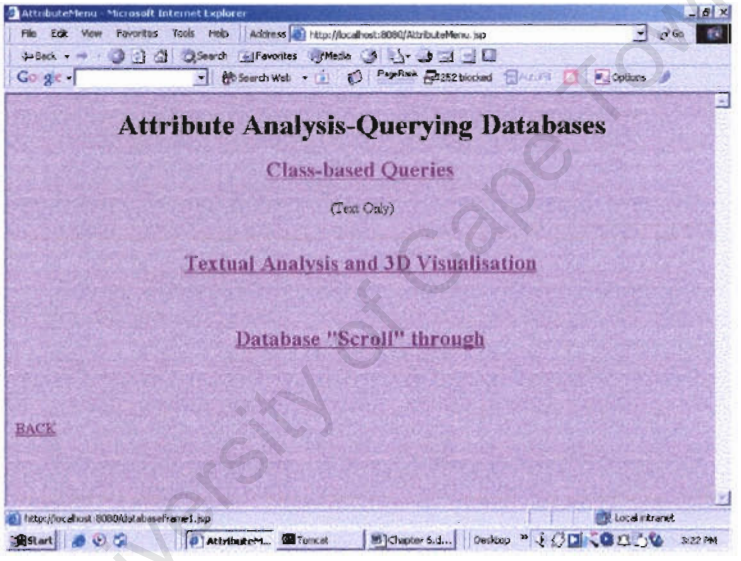


Figure 6.8. Attribute analysis menu page.

Class based queries: The user searches objects that meet specific conditions. In this case, the object has to belong to a certain class. The GUI for this operation is made up of two frames(Figure 6.9,below). In one frame, the user is presented with a drop down menu to allow him to choose from pre-determined object classes found within an urban environment. By avoiding requiring the user to type-in classes, this makes the application's use friendlier to non-experts. The right frame presents the textual results to the user.

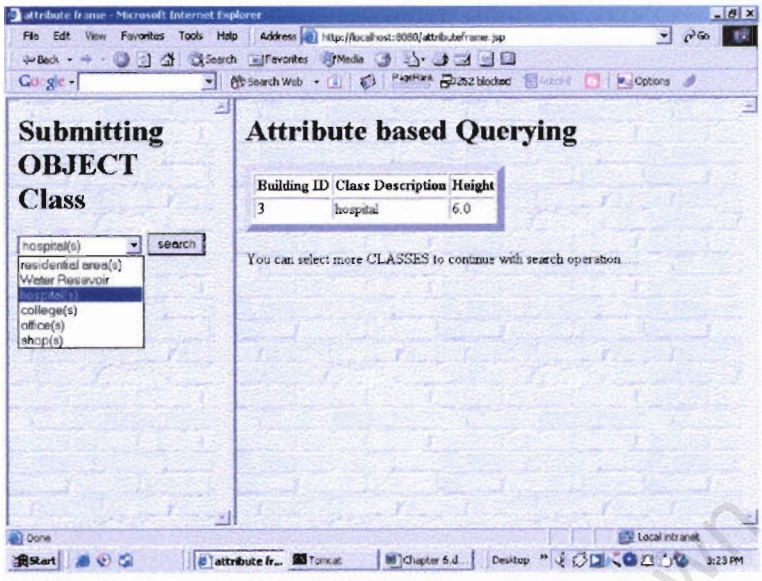


Figure 6.9. Showing the drop-down menu and attribute results

Textual analysis and 3-D visualisation: This GUI presents a way to extract information about a particular object. The user in this will have prior knowledge of the object identity (id-code). The user has at his disposal a two-section frame. On the left frame the user is invited to enter the object or building id and after searching the database, the search results are presented in the right frame(see Figure 6.10a). The User is given the option to go ahead and visualise the object with the attributes generated in the previous by pressing-on the visualisation control button provided. The search (or selected) objected is highlighted red (Figure 6.10b).

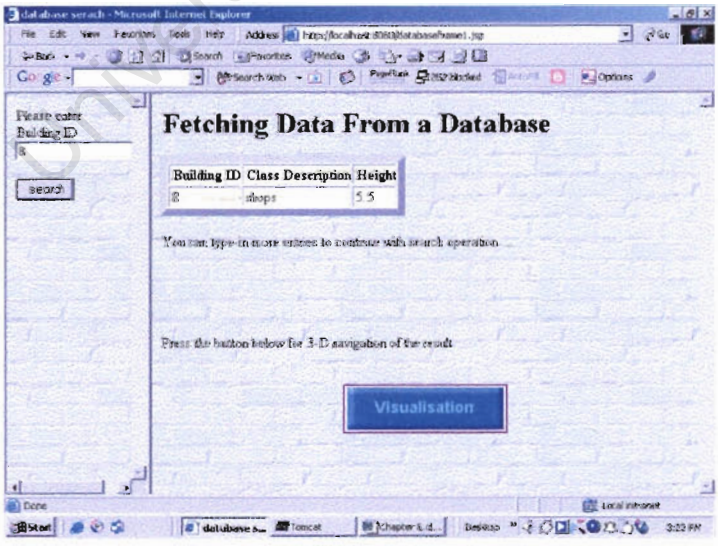


Figure 6.10a. object attribute data searches

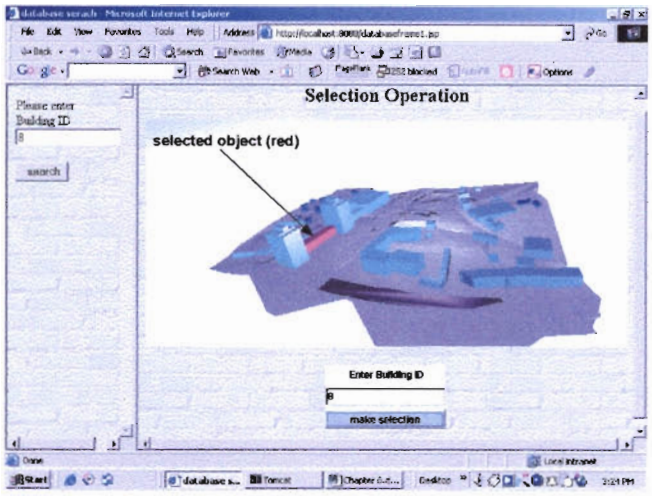


Figure 6.10b. Visual display of object selected.

Database "Scroll" Through: Database base scroll is useful to people who are visiting the application for the first time with no prior knowledge of the objects that are in the system. The user is provided with a single frame interface embedded with two control scroll buttons, previous and forward (Figure 6.11). By pressing on these control buttons the user is send queries to the remote database to see the objects ids and attribute information.

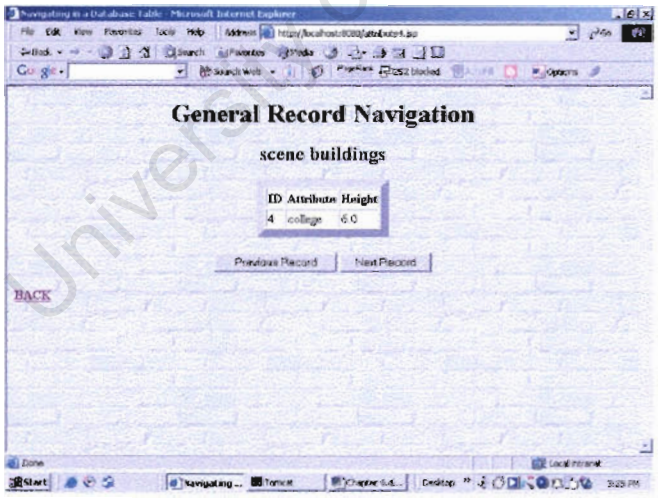


Figure 6.11. Database navigation

6.3.2 Neighbourhood Queries

One of the most important queries in Geo-information systems is that of neighbour determination. The application has two options to neighbour operations (Figure 6.12). General neighbour results

in the neighbours around an object and common Face analysis provides the common wall attributes to the abutting objects.

Figure 6.12. Neighbour search options

General neighbour search:

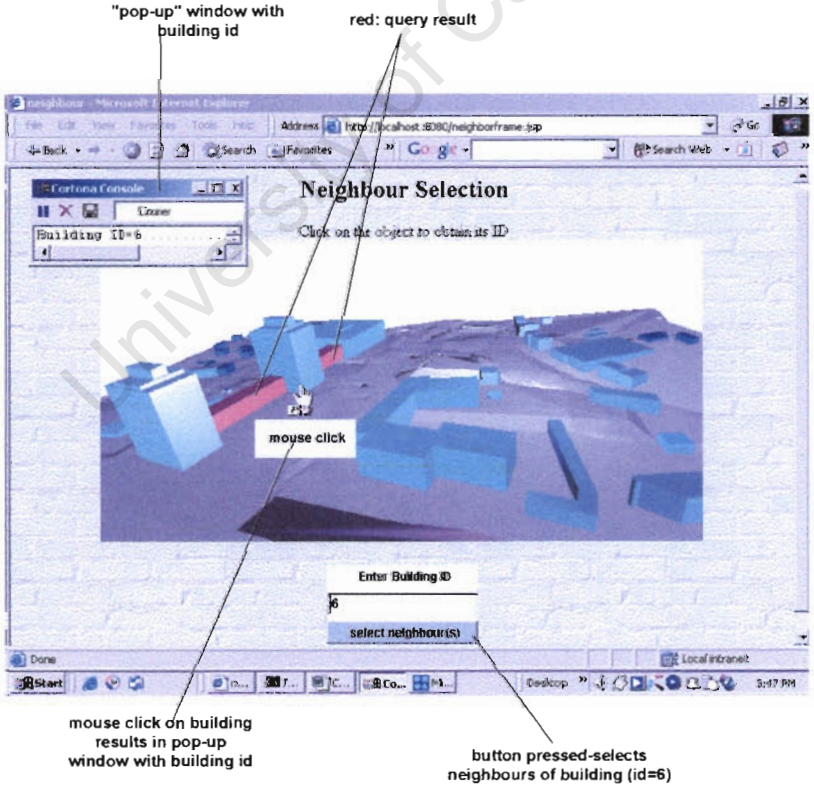


Figure 6.13. Illustrating the neighbourhood operation.

In the GUI provided in Figure 6.13 above, the Touch Sensor node senses when the cursor is over an object and when the mouse button is pressed, reports the ***id*** of the model. The information comes to the user through the pop-up window. Equipped with the object ***id***, the user can enter the id into the java-applet provided to submit the neighbourhood query. The results are visual; the neighbours to the object whose id is known are turned to the selection colour, red.

6.3.3 Common faces query

Firstly the user has to provide the building id which has known neighbours around it. The search operation as shown in Figure 6.14a allows for the retrieval of the common face(s) which are being shared with the object neighbours. The user then notes the common face ids. The applet interface in Figure 6.14b allows entry of the common face ids, which are then visually displayed in the VRML-generated 3-D model. In this scene, the buildings are deliberately of low opacity to allow the user to "see" through the buildings at the common walls (red).

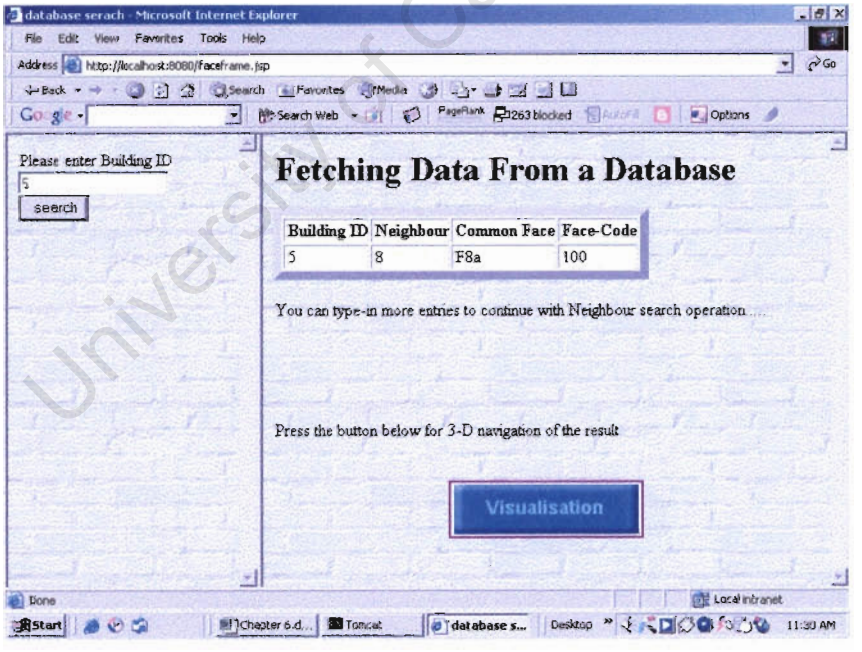


Figure 6.14a. Searching for the common wall id.

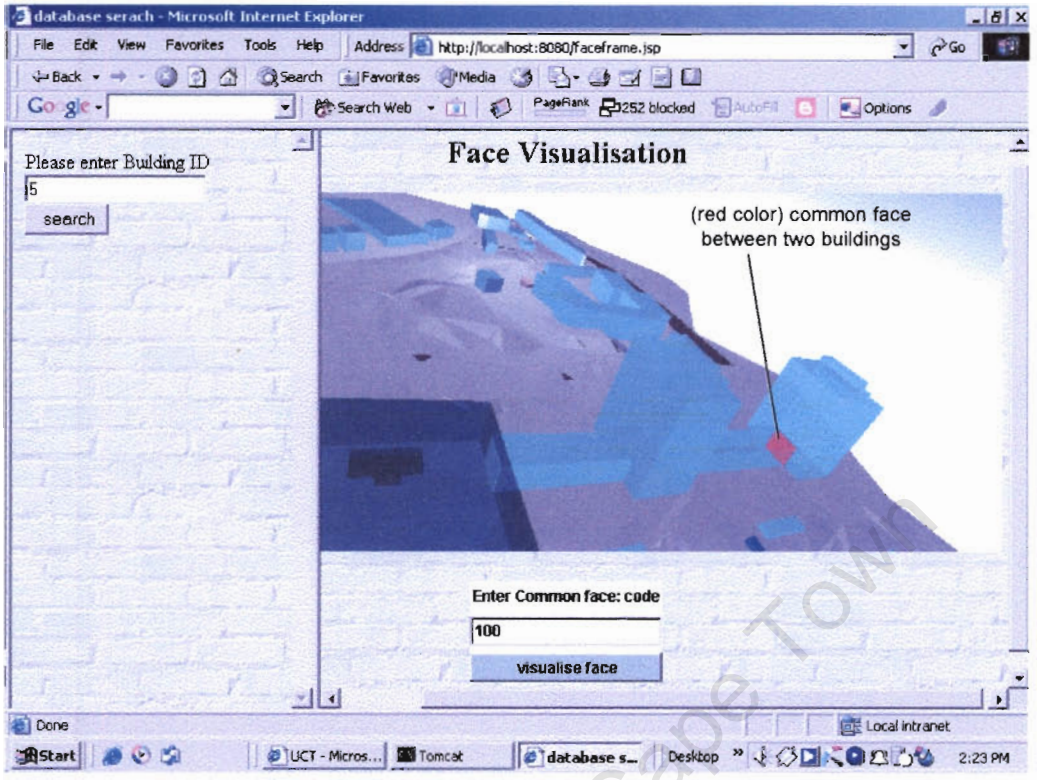


Figure 6.14b. Showing common face queried between two adjacent buildings. Note that the Buildings have been rendered transparent so that we could "see" through the walls.

6.3.4 Visibility Analysis

Theoretically, visibility analysis requires complex 3-D intersection algorithms between the line of vision and the faces forming the objects in the range of the line. Here, the user is presented with a simple solution based on a visual inspection of the actual path of a traversing line between two points. The line of vision is drawn in a VRML world and the user can observe the points of disturbance. Figure 6.15 illustrates this idea. The user enters the 3-D co-ordinates for the two points of the proposed line of sight into the applet interface. The provided "generate visibility line" control line will cause the creation of the visibility line from the entered co-ordinates.

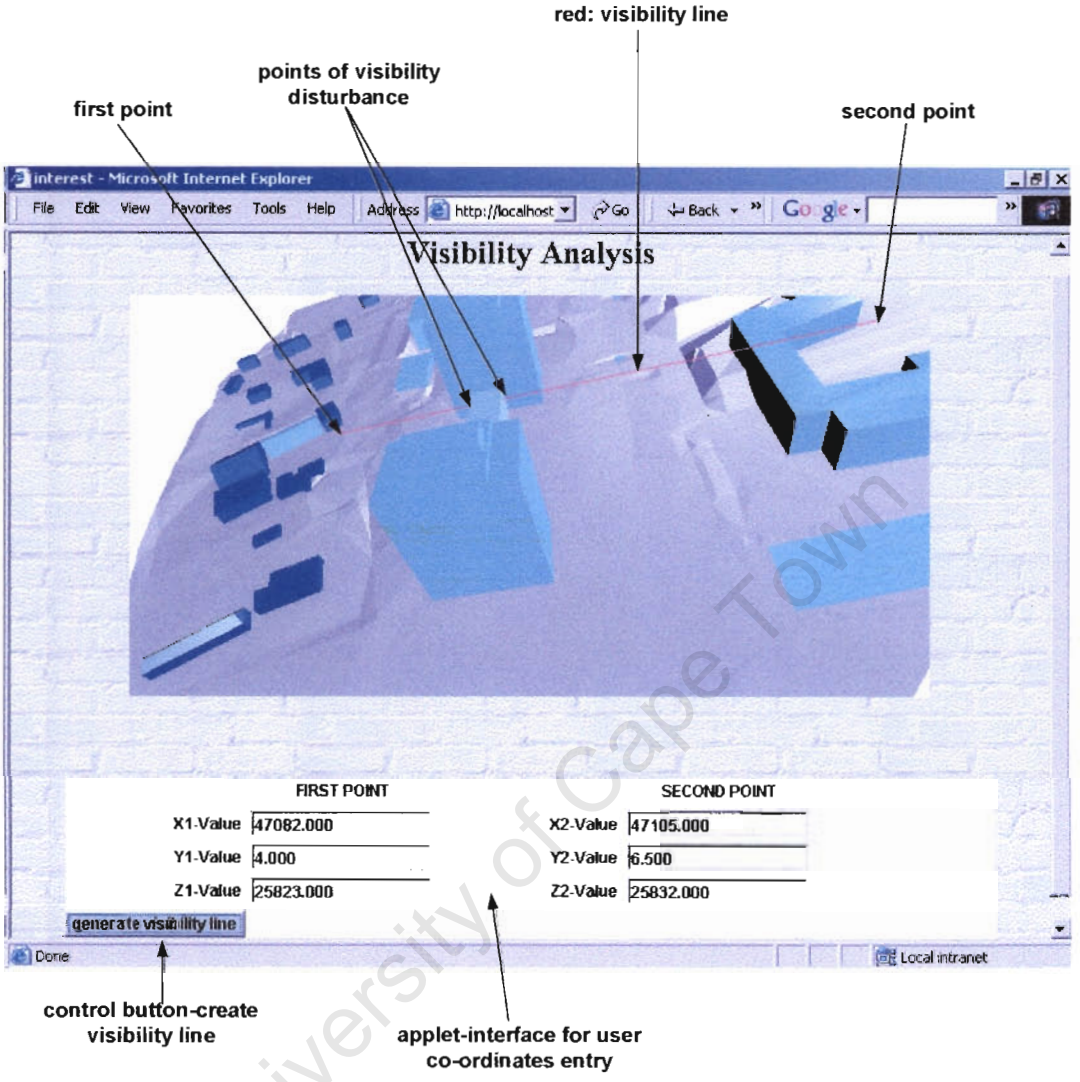


Figure 6.15. illustrating visibility analysis concept.

6.3.4 Buffer Analysis

As is the case with visibility operations, buffer analysis would require complicated algorithms. Again, this application presents a easy way out through the use of iso-surfaces generated from distance fields. In this case, the user types in a buffer distance in the "pop-up" that is provided and on pressing the key-board "enter" button, a buffer around the specified object is generated (see Figure 6.16, below)

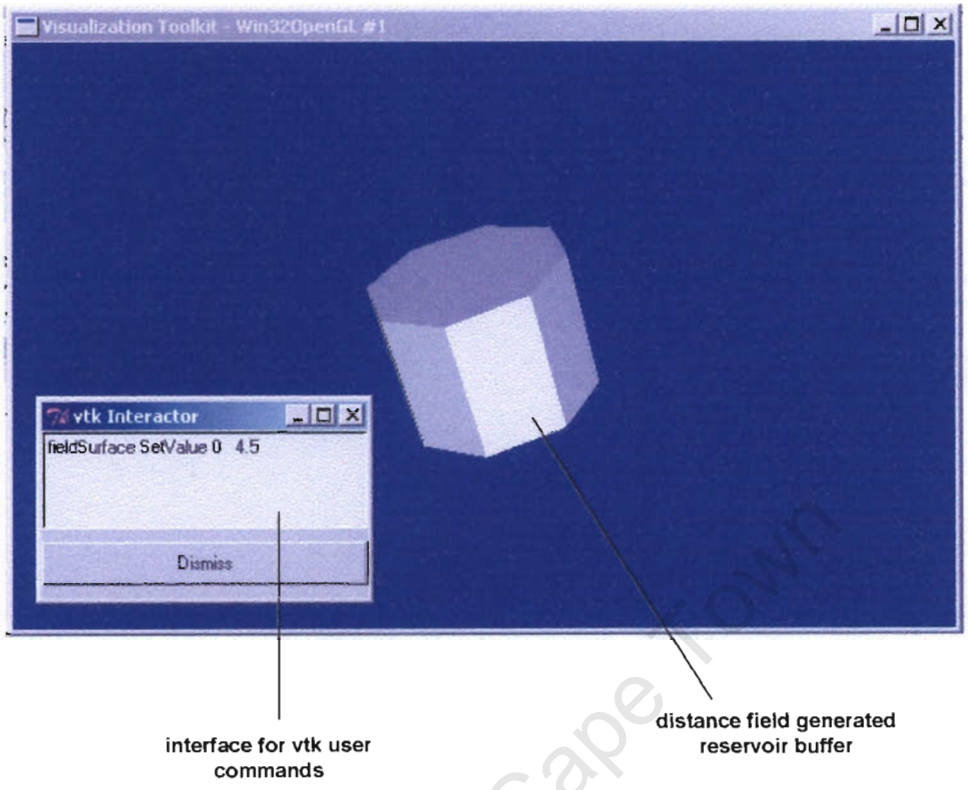


Figure 6.16. Buffer generated around a 3-D reservoir model.

Chapter 7
Evaluation

This Chapter focuses on the evaluation of the proposed H3-D representation and its implementation environment.

In general, an evaluation can be defined as the process of determining merit, worth, or significance. However, there are also common synonyms for the terms in this definition: "quality" is often used instead of "merit," "value" instead of "worth," and "importance" instead of "significance." (Scriven, 1991): The approach taken in this research builds up an "evaluation tree" (see Figure 7.1, below). The leaf nodes of the tree represent the lowest level (or component of the research) where evaluation will be implemented.

Directly linked to the evaluation tree is the criteria table,(see Table 7.1 below). Criteria, is broadly defined by Scriven(,1991) as " the characteristics of the evaluated entity that are of interest." The criteria definition in this research is based on the following:

1. qualitative criteria (characteristics that cannot be assigned a value directly and require further decomposition to which the set of questions will be applied successively until specific criteria are obtained) and
2. Quantitative criteria (characteristics that can be assigned a value directly using a particular data-gathering technique).

Once criteria are set, the yardstick has to be stated. Depending on the discipline there are different types of yardsticks (prescriptive standards, descriptive narration, etc.). All yardsticks must contain the specifications, requirements, descriptions, or values for each criterion considered. Whenever applicable, the yardstick must contain threshold values to indicate the minimum value for each criterion to be reached for a positive evaluation (Bergen,1995).

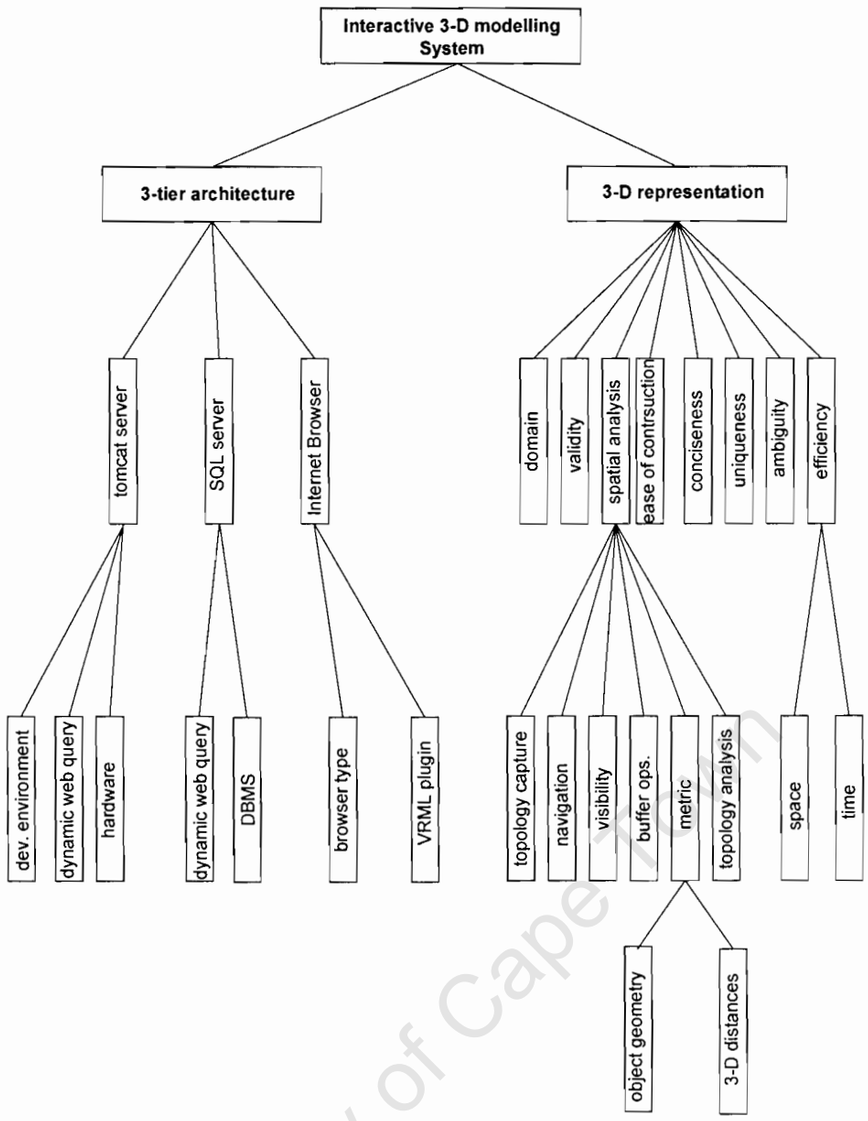


Figure 7.1. Criteria tree: decomposition of the interactive 3-D modelling system

This research considers both descriptive narration and prescriptive standards

Evaluation Criteria				Yard Stick
3D representation	Domain			3-D buildings in an urban environment, Linear features and point features.
	Validity			Valid under Euler's Equation for the B-rep component.
	Spatial Analysis	Topology capture		Winged edge data structure, 3D FDS
		Topological analysis		Winged edge and 3D FDS
		visibility		Prototype testing through entry of data
		Buffer operations		Proof of existence as this has not been implemented at all in other representations.
		Metric operations	Object geometry	Winged edge
			3-D distances	Proof of existence of algorithms
		Navigation		Proof of existence of tools
	Ease of construction			Winged edge, , Spatial partitioning and 3-D FDS
	conciseness			Winged Edge, Spatial partitioning, 3-D FDS
	uniqueness			Proof of existence
	ambiguity			Proof of existence
	efficiency	Space		Winged edge,3-D FDS, SSM and UDM
		Time		Winged edge,3-D FDS, SSM and UDM
3-tier architecture	Tomcat	Development environment		Should allow for Java, JSP, HTML ,Multi-threading
		Dynamic web query		Allow for Database queries , JSP, HTML
		Hardware/Memory		Standard Pentium 1.5Mbytes Hard disk memory
	SQL server	Dynamic web query		Interactive remote database querying .Open to remote requests through the use of ODBC and JDBC
		DBMS		Relational Database Support and SQL support
	Browser (client)	Web-browser type		Netscape or Internet explorer
		VRML plug-in support		Support EAI

Table 7.1. Defining the yard sticks for the evaluation criteria

7.1 Evaluation Process

The evaluation process is carried out systematically according to the structure of the evaluation tree.

7.1.1 Evaluation process: 3-tier architecture

The evaluation process for the 3-tier architecture is broken down into the sub-processes for: tomcat, SQL server and the web-browser (client).

7.1.1.1 Tomcat

The evaluation of the tomcat server is split into three components:

Server Development environment: The server accommodated a range of programmes developed in VRML (for 3-D model), SQL for Database queries, JSP for interactive programming, HTML generated responses from the SQL server and VTK for 3-D modelling of distance fields (see Chapter 6: Figure 6.5). This tolerance in a diverse of applications being launched from it ensured that the prototype was developed from a combination of off-the-shelf softwares without any deployment limitations.

Dynamic Web query: The Apache server ensured interactivity of displayed results formulated and ensured requests from the user through the use of JavaServer Pages. Integrated with the server is the tomcat container which is the engine for generating servlets from JSPs . By hosting the VRML and VTK files, the server allowed for 3-D interactive visualisation of the scenes called in by the web-browsers

Hardware and Memory : The requirements for available hard disk space of 1.5Mb, processor: at least a Pentium 486DX (233MHz) and RAM: at least 16Mb are easily met by most machines on the market today. As of year-end 2003, the entry level machine on the market was a Pentium IV i.e. processor (at least 2.4 GHz), Ram 512Mb and Hard disk space 40Gb. So the server can easily been installed onto a basic machine.

7.1.1.2 SQL server

The SQL server evaluation is given below:

Dynamic web query: The SQL server allows the user to specify queries in his browser using SQL syntax, and the data is returned as an XML document. The XML format provides a flexible and adaptable way of information delivery to different clients. The multi-threading functionality of Windows 2000, imply that many clients can access the database simultaneously.

Most of the Internet based applications (applets and servlets) use java code. By using the JDBC-ODBC bridge driver, the SQL enables the request from these Internet based applications to be interactively processed.

DBMS: Microsoft SQL Server 2000 is a relational database management system (RDBMS), which serves to provide data storage and management services. Microsoft SQL Server 2000 fulfils the research requirements of accepting and executing remote client requests. The client could easily add, modify and even delete the Server stored data in the tables (i.e. using the SQL syntax UPDATE, SELECT and DELETE statements). The relational modelling approach, is the implementation data model for the H3-D representation, and the Microsoft Server 2000 provided all the required functionalities.

7.1.1.3 Browser (client):

The evaluation of the browser (i.e. the client) is split into two main components:

Web-browser type: The user can interact with the H3-D generated models through Microsoft Internet explorer or Netscape explorer. It should be noted that java-scripting, which facilitates interactive visualisation, is supported in Netscape and not Internet explorer, hence, JSP based pages and applets were used, which meant that the developed prototype could be run in both browsers giving the same expected results.

VRML plug-in support:
The used browsers supported the tested plug-ins, Cortona, Cosmo Player and Blaxxun. Cortona plug-in was settled for because of it support for EAI operations in both browsers.

7.1.2 Evaluation process: 3-D representation

The evaluation of the 3-D modelling is split into the following:

7.1.2.1 Domain

The H-3D rep is capable of modelling 3-D urban entities, surfaces, linear and point features. A critical analysis of theH-3D logical structure (see Chapter 5: Figure 5.20) provides the following:

- 3D Feature modelling: This is possible because of the existence of the body feature entity type in the model. The relationships for this modelling, starting from the node (which has the geometrical co-ordinates) are: ***node-arc-face-body***.
- Surface feature modelling: The surface feature type ensures simple surfaces are modelled, and the DTM special feature allows specified algorithms to use the node data in generating digital terrain models. For surface feature, the relationships are: ***node-arc-face-surface***, and for the DTM, ***DTM-node***.
- Linear feature modelling. The relationships ***node-arc-line*** facilitates line feature modelling.
- To ensure non-manifold feature modelling, the arc feature is expected to have at least three face elements incident on it. This is made possible by the many to many relationship between arc and face primitives (***arc-face***) .

7.1.2.2 Validity

The validity evaluation is carried out separately on the B-rep component and distance field component.

B-rep component: With all B-reps, the primitives have to satisfy a number of conditions (see chapter 3):

- Each face must have at least 3 arcs(Figure 7.2): This is fulfilled by the following relationship in the model:

Figure 7.2. Extract of an E-R showing face-arc validity condition

The cardinality 3..*N* imply that the minimum number of arcs linked to any face is 3 and there is no limit on the upper bound.

- Each edge will have at least two nodes and the E-R extract below (Figure 7.3) illustrates this validity condition.

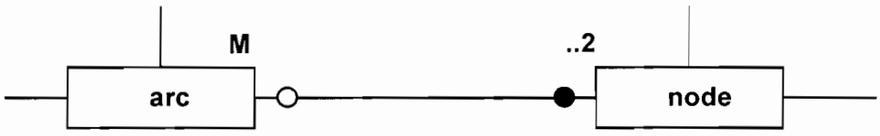


Figure 7.3. Extract of an E-R showing arc-anode validity condition

The ..2 cardinality indicates that the maximum number of nodes per arc is 2.

- There is no overlap between point features in 3-D space (E^3) (i.e. no two point features are equal).

  This condition if fulfilled by the fact that recursive relationships on the node entity type and on the point entity type have not been modelled. This would result in a point feature being "equal" to another point feature and similarly the same would happen on node features (see Figure 7.4, below). Again there is a strict one-one relationship between a point and node.

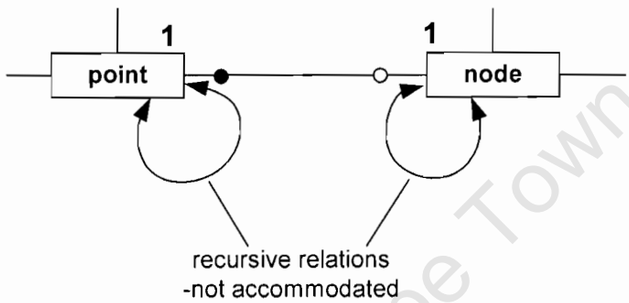


Figure 7.4: Illustrating point-node validity condition

- Topological validity: For topological validity, B-reps polyhedra satisfy Euler-Poincare formula (see Laurini, 1996). The H3-D expresses this invariant relationship among the number of vertices, edges and faces of a simple polyhedron as show below:

  Euler's formula: $v - e + f - L = 2(S - G)$

  where

  v :the number of vertices

  e :the number of edges (i.e. arcs)

  f :the number of faces

  L :number of Loops

S : the number of *shells*. A shell is an internal void of a solid. A shell is bounded by a 2-manifold surface. Note that the solid itself is counted as a shell. Therefore, the value for S is at least 1.

G: the number of holes that penetrate the solid (through holes).

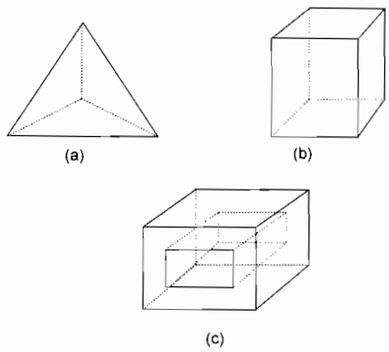


Figure 7.5: Polyhedra for topological validity

The following H3-D skeleton tables are used in deriving the values used in the Euler-Poincare Formula:

Node *(Node-id,……..)*

Arc*(Arc-id,……..)*

Face*(Face-id,……..)*

Body*(Body-id,……,L,S,G.)*

- Evaluation using object (Figure 7.5a):

The H3-D will record: ***v***=4, ***e***=6, ***f***=4 and ***S***=1, ***L***=0, ***G***=0

Therefore $v - e + f - L = 4 - 6 + 4 - 0 = 2$ and $2(S - G) = 2(1 - 0) = 2$

Proving that the representation generates a topologically valid model

- Evaluation using object (Figure 7.5b):

The H3-D will record: ***v***=8, ***e***=12, ***f***=6 and ***S***=1, ***L***=0, ***G***=0

Therefore $v - e + f - L = 8 - 12 + 6 - 0 = 2$ and $2(S - G) = 2(1 - 0) = 2$

This again proves that H3-D generates a topologically valid model.

- Evaluation using object (Figure 7.5c):

The H3-D will record: ***v***=16, ***e***=24, ***f***=10 and ***S***=1, ***L***=2, ***G***=1

Therefore $v - e + f - L = 16 - 24 + 10 - 2 = 0$ and $2(S - G) = 2(1 - 1) = 0$

This results in a topologically valid model.

As expected, the B-rep component of the H3-D generates topologically valid models.

7.1.2.3 Spatial Analysis

Topological analysis: The H3-D provides a number of analysis operations derived mainly from its Entity-relationship structure. The H3-D is evaluated for the topological queries ***T1-T10*** (see appendix B). This analysis evaluation results in the H3-D implementing all the queries.

Visibility analysis: The 3-D model generated form the H3-D is capable of implementing the visibility analysis. This is mainly possible due to the fact that the EAI is supported in the prototype set-up. The user can generate a visibility line within the 3-D scene and identify the obstructions. This is an operation that saves on computational costs as compared to the determination of line of sight equations and calculation of off-sets from planes and 3-D bodies.

Buffer operations: Buffer operations are achieved through the use of the DF component of the H3-D. Buffering in this case is a simple operation of specifying the distance to which one wishes to buffer using the surface generation algorithm (matching-cube). It has to be noted that the buffering operations has not been implemented in any known 3-D application, hence this evaluation is limited to proof-of-existence.

Navigation: Simple object search operations are made possible by the fact that the entire representation is modelled through the Entity-Relationship approach to which the fundamental link chain is the feature ID. So if one is given the feature ID, it is easy to move from one feature table to the other retrieving relevant attributes and information. The navigation is possible from a 3-D VRML generated scene to the database and as well as from the 3D VTK generated scene to retrieve attribute information from the database as well.

7.1.2.4 Metric operations (Object geometry & 3-D distances)

- Object geometry: Object geometry can easily be derived from the node table which has ***x,y,z*** tuples. Evaluation of this item is limited to "proof-of-existence" of the co-ordinates within the node entity type table. The algorithms for metric analysis have been deliberately left out as they have been well researched and established. This research would not add any further value to what has been implemented in other systems.
- The evaluation of 3-D distances is limited to the fact that a distance algorithm (see Appendix B) was developed. Again, all the operations (algorithms) implemented for metric analysis

operations have been done so for the first time. There is no known targets to compare them against. Proof of existence is sufficient in this instance.

7.1.3 Ease of construction

The H3-D is implemented in a distributed environment made up of a number of different software systems. The Database component is implemented in Microsoft Server 2000, the application server which is basically a java environment runs the application logic and the client VRML and VTK applications. Because of this set-up, the H3-D is not easy to implement in it current form.

7.1.3 Conciseness

The H3-D representation can be described as verbose. It has the B-rep component made of a number of sub-components and the distance field component. Accompanying algorithms are linked to the Distance Field component (e.g. matching cube, interpolation, search algorithms).

7.1.4 Uniqueness

The uniqueness issue remains outstanding, as is the case with all B-reps. It is difficult to assign a particular resolution of surface decomposition. But because of the geo-referencing of models, H3-D ensures that the position will mean that a single model within the model domain.

7.1.4 Ambiguity

The manner of data acquisition and surface subdivisions are different for each object. The probability of having two models with exactly the face definitions is therefore very low. This ensures unambiguity in the modelling.

7.1.5 Efficiency

An important question to ask in this research is " how efficient is the developed H3-D?". Efficiency covers, among other issues, CPU time usage and memory usage which are the focus in this evaluation.

Space: Space or memory complexity is investigated by counting analysis. The number of records obtained per specified primitive (problem size-N) is compared between an established representation and H3-D representation. Objects given in Figure 7.6 are used in the analysis. The Wing-edge, UDM, SSM and 3D FDS are considered the yardsticks. Plots of records against geometric elements (i.e. primitives) are developed for all the reps. Functions for the "best fit" curves are generated (from least squares regression analysis: see Laurin et al. 1996) and the correlation co-efficient (r^2) (i.e. a measure of how well the predicted values from a forecast model "fit" with the real-life data. The correlation coefficient is a number between 0 and 1. If there is no relationship between the predicted values and the actual values the correlation coefficient is 0 or very A perfect fit gives a coefficient of 1.0**).**

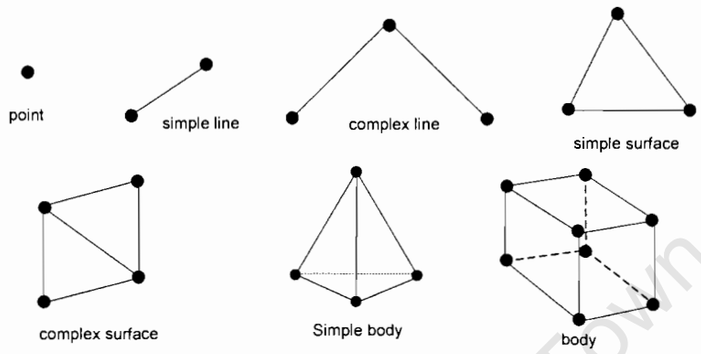
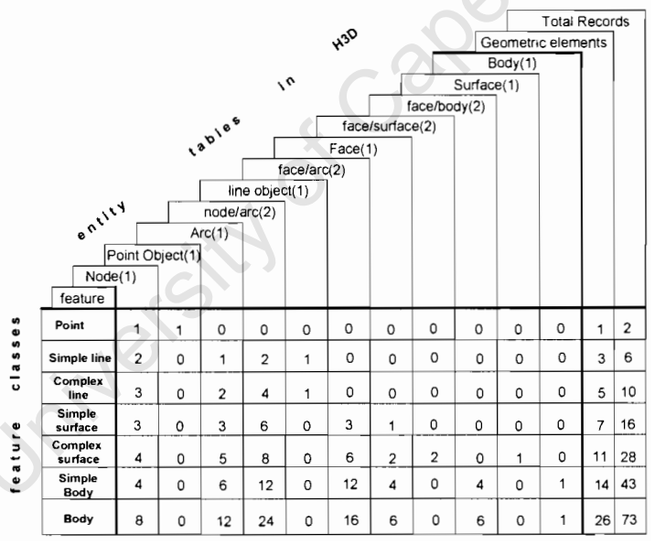


Figure 7.6. Various features for analysing space complexity



feature classes	Node(1)	Point Object(1)	Arc(1)	node/arc(2)	line object(1)	face/arc(2)	Face(1)	face/surface(2)	face/body(2)	Surface(1)	Body(1)	Geometric elements	Total Records
Point	1	1	0	0	0	0	0	0	0	0	0	1	2
Simple line	2	0	1	2	1	0	0	0	0	0	0	3	6
Complex line	3	0	2	4	1	0	0	0	0	0	0	5	10
Simple surface	3	0	3	6	0	3	1	0	0	0	0	7	16
Complex surface	4	0	5	8	0	6	2	2	0	1	0	11	28
Simple Body	4	0	6	12	0	12	4	0	4	0	1	14	43
Body	8	0	12	24	0	16	6	0	6	0	1	26	73

Table 7.2. Primitives and data records for the H3-D representation

The table values (Table 7.2) are derived directly form the logical data structure of the H3-D representation (see Chapter 5:).For example, a simple node will have 1 record in the node entity table, and a single record in the point entity table, resulting in a total of 2 records. The geometric elements refer to the primitives (i.e. node, arcs and faces) making up an object. The number in the parenthesis of every entity type indicates the minimum number of attributes per entry. Therefore the records per primitive entry, are obtained from multiplying the number of entries by the parenthesis number.

feature classes	feature	vertex	incident edge	Vert-no.	face	incident edge	face No.	edge No.	vertices	start	end	faces	left	right	left traverse	succeding edge	preceeding edge	right traverse	succeding edge	preceeding edge	Geometric Elements	Total Records
												E D G E										
Point		1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Simple line		2	0	2	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	3	9
Complex line		3	2	4	0	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	5	23
Simple surface		3	3	1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	7	35
Complex surface		4	4	2	2	5	5	5	2	0	5	5	5	5	5	5	5	5	5	5	11	57
Simple Body		4	4	4	4	6	6	6	8	6	6	6	6	6	6	6	6	6	6	6	14	70
Body		8	8	6	6	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	28	136

Table 7.3. Primitives and records for the Wing-edged data representation

The winged –edge values are obtained from the logical design tables (see Appendix A).

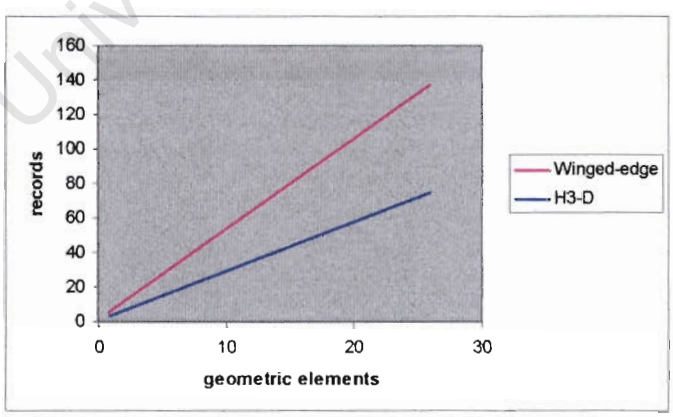


Figure 7.7. Plot of records against Geometric elements for the Winged-edge and H3-D reps

A plot of records(Figure 7.7, above) against primitives is drawn for each of the two representations. The functions that describe the "best-fit" lines are:

Winged edged:

$R_{Wing} = 5.2549N$ and $r^2 = 0.9755$

H3-D: $R_{H3D} = 2.849N$ and $r^2 = 0.9744$

The space complexities defined in terms of the functions for the two representations are given by:

Winged edged: $O(5.2549N)$

H3-D: $O(2.849N)$

And ignoring the constants and lower order terms:

Winged-edge: $O(N)$

H3-D: $O(N)$

Note that the lower order terms, co-efficient and constants affect the growth rate when the problem sizes are small. The inference from the above analysis is that the H3-D (B-rep component) scales better than the winged edge with reference to space requirements for small problem sizes. As the problem size grows, the two reps scale equally. It should be noted that since the implementation design for the H3-D is a 3-tier architecture, the space under consideration on the database server. The H3-D representation needs further space on the application server for 3-D distance map and VRML 3D models.

SSM in tables

entity

feature

Node(2) =(10)

Point [A,B,T] =(5)

Text[2,D] =(13)

Line[T,B,A,D] =(13)

Face(4)

Surface[T,B,A,D] =(13)

Body[T,B,A,D] =(13)

Composite Ob(T,B,A,D) =(13)

Geometric elements

Total Records

feature classes

Point	1x2	10	0	0	0	0	0	13	1	25
Simple line	2x2	0	0	13	0	0	0	13	3	30
Complex line	3x2	0	0	13	0	0	0	13	5	32
Simple surface	3x2	0	5	0	4	13	0	13	7	41
Complex surface	4x2	0	5	0	8	13	0	13	11	47
Simple Body	4x2	0	5	0	16	0	13	13	14	55
Body	8x2	0	5	0	24	0	13	13	26	63

A=5 records B=3 records D=3 records T=2 records

Table 7.4. Primitives and records for the SSM representation

Again, the parenthesis indicates the records per each entry under an entity type.

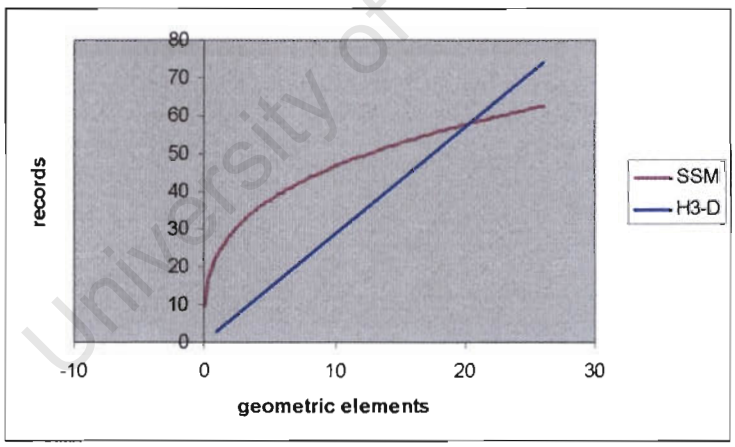


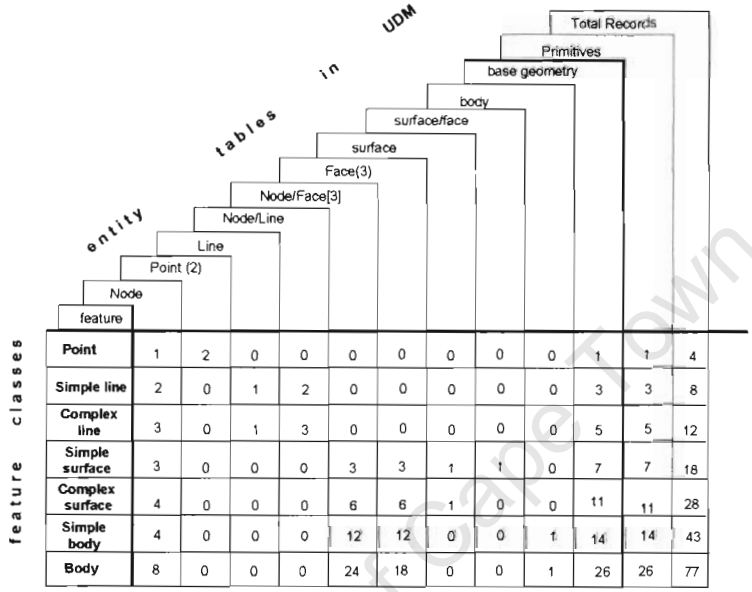
Figure 7.8. Plot of records against Geometric elements for the SSM and H3-D representations

SSM: $R_{SSM} = 22.838N^{0.3089}$ and $r^2 = 0.929$

H3D: $R_{H3D} = 2.849N$ and $r^2 = 0.9744$

There is a marginal difference between the two representations in terms of the storage costs, with the SSM scaling as $O(N^{0.3089})$ and the H3-D scaling as $O(N)$ after ignoring the lower order

terms. Because the SSM lacks the arc and edge primitives, one would expect it to conserve much more space, however, the drawback emanate from the fact that it is intended for visualisation and to some extent for animation purposes as it stores an objects' behaviour, thematic attributes, and geometric description and this is directed towards visualisation with VRML players, and in the process, it raises the memory requirements of the SSM (see Zlatanova, 2000).



feature classes	feature	Node	Point (2)	Line	Node/Line	Node/Face[3]	Face(3)	surface	surface/face	body	base geometry	Primitives	Total Records
Point	1	2	0	0	0	0	0	0	0	1	1	4
Simple line	2	0	1	2	0	0	0	0	0	3	3	8
Complex line	3	0	1	3	0	0	0	0	0	5	5	12
Simple surface	3	0	0	0	3	3	1	1	0	7	7	18
Complex surface	4	0	0	0	6	6	1	0	0	11	11	28
Simple body	4	0	0	0	12	12	0	0	1	14	14	43
Body	8	0	0	0	24	18	0	0	1	26	26	77

Table 7.5. Primitives and records for the UDM representation

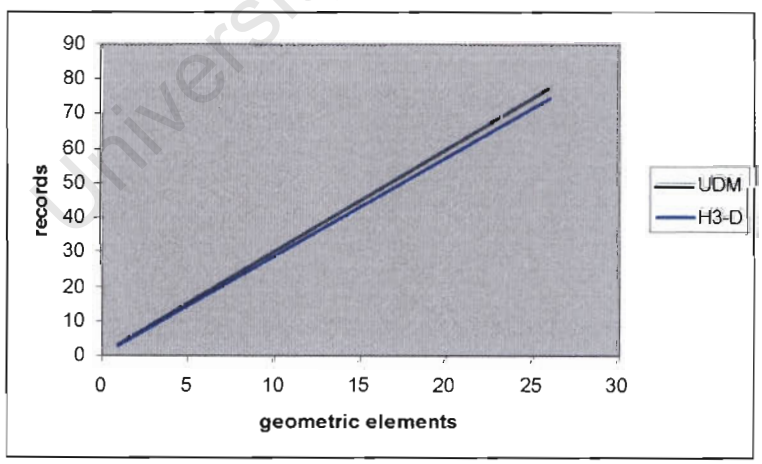


Figure 7.9. Plot of records against Geometric elements for the UDM and H3-D representations

UDM: $R_{UDM} = 2.982N$ and $r^2 = 0.9905$

H3D: $R_{H3D} = 2.849N$ and $r^2 = 0.9744$

The storage costs for the H3D are marginally higher than UDM, and this holds for small 3-D scenes. As the problem size becomes bigger, the two representations fare equally, i.e. Order $O(N)$. Although the UDM lacks the arc primitive, the representation has a geometry feature and specifically the "left" and "right" body/face attributes, thus increasing the number of its records.

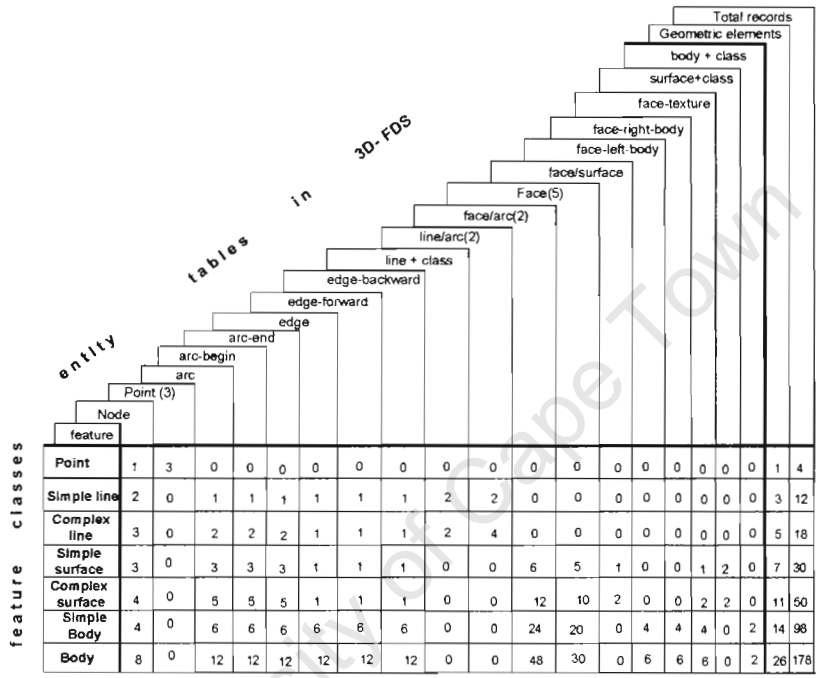


Table 7.6. Primitives and records for the 3D FDS representation

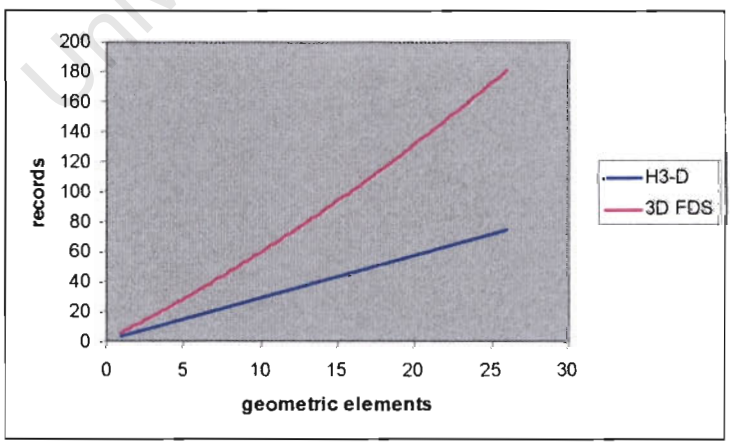


Figure 7.10. Plot of records against Geometric elements for the 3D FDS and H3-D representations

3D FDS:

$R_{3D\,FDS} = 0.06777N^2 + 5.1915N$ and $r^2 = 0.984$

H3D: $R_{H3D} = 2.849N$ and $r^2 = 0.9744$

The 3D FDS fares badly compared to the H3-D representation, i.e. Order $O(N^2)$. This is expected because of the presence of the edge primitive, the node-in-body singularity, shape information and the "left" and "right" attribute values for the face. The logical entity-relationship model gives rise to more data tables as well. The representation has a far more attributes for the entity types than the H3-D.

Time: Exactly how much time does each component of a program require? To do this in terms physical clock cycles, we must know exactly how the statements are translated into machine language and how many nanoseconds each machine instruction takes. Since this depends up particular machines and or systems, we shall adapt a less stringent policy (as is the case with memory evaluations) and just count the primitive operations that are apparent from the program algorithms. In all the cases analysed below, the relational modelling technique is assumed as the database design approach.

A comparison of algorithms for the Wing-edged, UDM, SSM, 3D FDS and H3-D is undertaken for a number of spatial queries(see Appendix B).

Table 7.7 reveals that the H3-D(B-rep component) and 3D FDS representations can have a direct implementation of all the queries from the sample size and the UDM and SSM could only manage 50% of the queries. The analysis reveals that the gains in efficiency by the UDM and SSM are mainly because of the non-availability of the arc attribute and this however has a cost i.e. reduction in query capabilities. The comparison between the H3-D and 3D FDS indicates that the H3-D is more efficient (H3-D has 4 least time complexities compared to 3d FDS's 2)

The time complexity for the H3D fares well throughout, exhibiting least complexities for the queries ***T1-T3***,***T5***,***T6***,***T8*** and ***T9.*** For ***T4***,***T7***and ***T10***, it is comes second best.

		REPRESENTATIONS				
		Winged-edge	H3-D	3D FDS	SSM	UDM
QUERIES	T1	$NK+6K$	$v+6K$ *	$2v+12K$	—	—
	T2	$2K$ *	$N+4K$	$N+4K$	—	—
	T3	$5K$ *	$N+4K$	$N+4K$	—	—
	T4	$2NK+K$	$5N+12K$	$9N+20K$	$M+4K$ *	$M+4K$ *
	T5	—	$M+2K$	$2K$ *	$M+2K$	$2K$ *
	T6	—	$b+6K$ *	$2b+6K$	$b+6K$ *	$2b+6K$
	T7	—	$30N+b+78K$	$54N+2b+126K$	$b+6M+30K$ *	$2b+6M+30K$
	T8	—	$6N+b+30K$ *	$6N+2b+30K$	—	—
	T9	—	$N+4K$ *	$N+4K$ *	—	—
	T10	—	$5N+12K$	$9N+20K$	$N+2K$ *	$N+2K$ *

* Indicating the least time complexity

— query, diffuicult to implement

Table 7.7. Summary of time complexities for Queries T1-T10

Chapter 8
Conclusions and further research

The research approach taken in this work includes conceptual designs, logical designs and physical designs. The main area of this research has been the development of a hybrid representation for 3-D modelling. This hybrid approach has not been previously undertaken within the area of urban modelling. As a result, review work has been limited to single format representations. Faint references have been made to the use of distance fields in spatial data modelling, but no research efforts have been reported. This research has introduced some concepts in the use of Distance fields in urban modelling. The research goes further and introduces some adaptations and design of some algorithms within the distance fields. Although the research efforts that have been undertaken in 3-D representation modelling have been immense, the one single biggest weakness has been the lack of the ability to fully define the implementation environment, an issue addressed by this research. This weaknesses is perceived as coming from the fact that research of this nature links several fields namely GIS, Computer Graphics, Data modelling and Distributed Computing. As such the solution to tackling research of this nature is by carrying it out across these different professions, an approach that resulted in this research being carried out in the Departments of Computer Science and Geomatics. The prototype is limited to a few key spatial queries and visualisation operations. This was mainly due the fact that the research currently under development makes use of "freely" available tools from the internet, and some of these tools are being developed (e.g. VTK related tools).

Urban managers, planners and the general users would benefit from this system in the following ways:

- Reduction in cost for set up: a PC and internet connection
- Enlarged scope of queries to be preformed, i.e. static and dynamic analysis operations
- Virtual reality navigation and exploration of 3-D worlds
- Sharing of information with remote users
- Operating the system without the need for knowledge of database design and system installation skills

8.1 Conclusions

The conclusions are reviewed under the three set objectives for this research:

Objective 1: To design a Hybrid 3-D representation .....

The research makes a thorough review of different categories of representations in chapter 3 of the thesis and comes up with the need to develop a hybrid of B-rep and Distance Field representations. A further review of the B-reps in chapter 5, highlights their weaknesses and suitability to participate in hybrid schemes. However, this extensive background review of representations provides the sound theoretical foundations on which the H3-D representation is developed. The H3-D primitives and object relationships are defined using set-theory. This approach makes it easier to map the objects and relationships into a database management system. hybrid representation is a fusion of distance field and boundary representation components. 3-D visualisation and spatial analysis operations are shared between the two components, exploiting the best features of both. The B-rep component implements the general "static" spatial queries, whereas the distance field is ideal for "interactive" querying. Whilst the data in DBMS is stored through the E-R logical design of the B-rep component, the distance field model component is generated on the "fly".A review of the possible distance field modelling techniques in chapter 5 resulted in the ***sDF*** approach being adopted for this research. The notable characteristics of the new H3-D rep are:

- It can implement 3-D buffering, an issue that has evaded many 3-D rep research efforts
- The rep allows the user to interactively determine the distance of a point to the nearest point on an object (location distances).
- PMC (Point Membership Classification) and LMC (Line Membership Classification) queries are interactively executed.
- 3-D models are linked to an attribute database, thus allowing interactive querying of the attributes during navigation.
- The H3-D domain covers linear features, surfaces and bodies. The many-to-many (***M:N***) relationship between the face and arc entity types, ensures that the representation models manifold and non-manifold bodies. The cardinality between face and arc entity types ***M to 3...N*** (i.e. a many to many with a minimum of 3 entities on the arcs) will ensure that valid faces (minimum of 3 sides) are formed. The same can be said of the relationship

between the arc and nodes, ***M to ..2*** (i.e. two to many – with the constraint that two nodes define an arc).

- Neighbourhood queries have been implemented and interactive querying is possible. The H3-D could implement all the ***T1-T10*** neighbourhood queries from the sample set (see Chapter 7).
- The DTM feature ensures that there is a better approximation of topography within 3-D scenes.
- The single Distance Fields approach ensures that all urban features are modelled within the same distance field map. The feature attributes, assigned to the root cells within the hierarchical octree structure, ensures that object queries are efficiently implemented.

The representation's performance is evaluated in chapter 7 of the thesis and is summarised as:

- An analysis of the space costs reveals that the H3-D performs marginally better than the 3D FDS. It fares the same compared to the UDM and Winged-edge reps. The lack of the edge primitive within the SSM means it conserves marginally more space. However, momentary space requirements do go up as H3-D generates VTK files and VRML files which are stored on the application server. There is an option though to retain the generated files for reuse.
- Chapter 7 provides an analysis of program time complexity for H3-D and this is compared with four other representations. This analysis is based on algorithms for a sample set of queries, ***T1-T10***. The representation is more efficient in 4 of the 10 queries compared to the other reps (the highest percentage within table 7.7, see Chapter 7).
- The H3-D rep (along side the 3D FDS) provides simple algorithms for all the queries ***T1-T10***, a 100% efficient response to the sample set.

Objective 2: designing and adaptation of the distance field environment…..

The second objective was focussed on the generation of the DF environment that would allow for 3-D visualisation and spatial analysis of urban scenes.

The adaptive grid proposed in this research was be mapped through an octree hierarchical structure. In the traditional grid (see Frisken et. Al. 2000), the storage of distance field values was done for all the eight corners of a cell per given node. This approach has the obvious

disadvantage of redundancy in data storage values as some grid points are shared by nodes at the same level as well as being shared by the children and parents of the nodes. This research proposes a modified ADF which ensures that data redundancy is eliminated, thus requiring less storage space. The modified ADF approach stores approximately an eighth of the space occupied by the general ADF structure.

The other notable difference between the traditional approach and this research is that the blank nodes are stored more than once in the traditional approach whereas in the modified ADF, these are stored only once.

This research makes an in-depth analysis into two possible designs of developing the distance field maps, the ***sDF*** and ***mDF***. The strengths of ***sDF*** over the ***mDF*** are clearly indicated. The *sDF* conserves almost twice the amount of space as the *mDF* and on a similar scale, performs spatial analysis operations more efficiently (i.e. has a better time complexity).

Since the ***sDF*** uses a single DF map, it is conveniently the choice for this research as it allows analysis of features to be carried out within the same scenes, unlike the ***mDF*** which provides different layes for each and every feature mapped.

It has to be noted that the ***sDF*** is not a normal DF as it includes feature attributes and as a result, it has a set of new algorithms.

Objective 3:

…..design of 3-D spatial analysis algorithms for interactive analysis and prototype development

The first part of the objective dwelt on the design of the dynamic spatial analysis operations. A chosen set of queries is investigated for implementation by the ***DFs***. In justifying the use of ***DFs*** in the implementation of these queries, comparisons were made against B-reps and in the process, exposing the limitations found within the B-reps.

The memberships classifications, buffer and point location where the main analysis operations which where investigated for, in this research. The adapted and derived algorithms for B—reps

on PMCs clearly indicated the difficulties that are faced when the PMCs are carried out on non-regular and ill-defined shapes. This has been highlighted as the strength for ***DFs***.

The ***sDF*** approach maps all the features within the same map, there is no need to carry out tests in each and every distance map as is the case with the **mDFs** approach. In this instance, the time for the operation for PMC is constant with complexity O(*c*), where ***c*** is the constant overhead. This time is thus not affected by an increase in the number of features being modelled.

The line member determination approach for ***sDF*** is similar to the ***mDF*** approach above. The algorithm however includes the analysis of distance field object attributes. This distance is readily provided in a distance field map which only represents a single feature but the situation is different in ***mDFs***. In B-reps, the approach would involve a number of mathematical computations which would make interactive querying unattractive.

The point location based queries posed some challenges to the ***sDF*** scheme. Contrary to the approach of ***mDFs***, in this, a point of interest could be located within the field of a feature *A*, and queries made to a distant feature *B*. A nearest neighbour search algorithm was designed to facilitate this operation.

Buffering of point data is easily implemented in "2.5-D systems than in ***DFs***. The "sphere" surface obtained within the DFs enviromenets (i.e. ADFs) is not perfectly spherical. Line buffering is carried out through simple dinace off-setting, an operation well cuttered for by the marching cubes algorithm.

Body feature buffering remains a topical issue in 3D GIS. This research makes maximum use of the offsetting quality of distance fields in implementing 3-D buffer Distance buffering in ***mDFs*** creates a buffer field around each object, and in separate scenes. Relative analysis operation between objects cannot be carried out but buffer zones are of unlimited radius.

The research has seen two possible ways in which buffer operation are carried out (in ***sDF***). Firstly, the buffer can be generated around the entire objects scene simultaneously and this would simulate a query of the form "create a buffer 2meters around all features".

The second approach generates a buffer around a single feature which can overlap into neighbouring features. This functionality is ideal when one models (in 3-D) the spread of smoke (or gases) from a point source.

Location distance analysis is easily implemented through two main steps; application of the nearest search algorithm and Pythagoras algorithm. This approach presents a far much better solution than that of determining going through a number of computational steps in B-reps.

The H3-D representation was implemented in a distributed Internet environment, and a 3-tier architecture was chosen as the distribution design approach. 3-D data is quite voluminous and expensive to collect, and this design ensures the sharing of information and application functionalities across networks. This design ensured that the applications' "load" was shared across the three tiers, with the server handling the basic data, Apache tomcat handling the application logic and generated files, and the web browser providing the systems interface to the user. According to this design, system scaling can be effected on the application server. This is a definite advantage over traditional 2-tier systems in which the application load is shared between the client and the data server and changes are effected on the multitude of clients linked to the system. Because 3-tier design relieves the client of application functionalities, the developed application is faster than a comparable 2-tier architecture.

The 3-tier approach also allowed heterogeneous systems to be combined: an SQL server running on Windows 2000, an Apache application server, VTK files , VRML files, JSPs and HTML files. This conforms to the ideals of this research, i.e. the use of off-the shelf applications, which are heterogeneous in nature.

JavaServerPage programming was used as the web page development "language". This was a big improvement for distributed systems as they previously use CGIs which consume resources and are proprietary applications. The JSP promoted dynamic web querying of the remote database. On the client side, Cortona VRML plug-in ensured that 3-D scene visualisation was possible within a web browser. The system uses either Netscape or Microsoft Internet explorer as the client. The use of Cortona was important in that it supports the EAI in both Netscape and Microsoft environments. This is crucial in that remote users are not restricted to the type of web-browser.

Microsoft SQL server 2000 provided the DBMS for the research. It supports the entity-relational modelling technique. The server system has an interface that made database modelling attractive. Remote queries were well supported through the use of the ODBC-JDBC bridge. The server returned results in XML format which are easily transmitted and read by remote clients.

The application server chosen for the system was Apache-tomcat. The Tomcat container allowed for the conversion of JSP pages to servlets which transmitted requests to the remote database.

For 3-D distance field modelling, the system employed the VTK application. This application was originally developed for medical systems and more research is still being carried out. A VTK interface was loaded on the client side.

8.2 Further Research

This research covered a wide range of issues as it explored a new area of Urban modelling using Hybrid representations and in the process, a number of important issues need further research:

Hybrid development

1. Future research should focus on developing the hybrid representation in an entirely object oriented approach. This however, will be accompanied by the development of the O-O filters that would "read' the data sets from an objected oriented environment.
2. Because of the lack of complete spatial analysis tools within the ***DFs***, one would contemplate implementing the entire representation within the ***DFs***.
3. Data updates within the DF environment is an issue that needs attention. Most users are interested in knowing how they can graphically update their scenes. Further research into this issue would closely look at how this updates will be put back into the central DBMS, without violating the logical modelling rules.
4. Within the B-rep environment, some investigations have to be undertaken in areas such as the automatic detection of semantic consistency violation.
5. The area of data updating with the B-rep component needs further investigation. For example if a smaller building were to be erected abutting a bigger one, this would mean that there will be a common face between the two. For this to be properly captured, the existing face on the Bigger building will have to be subdivided into two or more faces to accommodate the face from the smaller building. Geometrically this can be done easily,

but there are table attributes that need modification as well. Hence, updating scenes in 3-D modelling is an issue that needs due consideration.

6. The research assumes data is provided in point coordinates that defines object surface. But more research needs to be undertaken to link the H3-D representation with the data acquisition techniques. This would result in the entity tables being populated during the acquisition process.

Distance Fields and Prototyping

1. Research in the area of Distance Field modelling area is being spearheaded by the medical profession. At the time of putting together this thesis, efforts were underway to develop VTK Java classes through the famous VTK open forum on the internet. A VTK plug-in in the same mode as the VRML plug-ins is not far off, and this again would allow modification of this research to allow for EAI, thus obtaining a "complete" system.
2. The LOD (Level of Display) tools have been well developed for VRML systems and these are yet to be fully implemented in VTK, to allow the user to "zoom" in and "out" and in the process, changing the amount of detail presented in the users window. Buffering is a simple offsetting exercise within the DFs modelling, but the buffered shape is much distorted compared to the original object shape. This is because the shape was generated with an adaptive grid in areas of fine detail, but this does not extend to areas of buffering because these are not known before hand. The interesting issue of research here, would be to interactively generate an adaptive mesh.
3. The database structure in the VTK system is relational, but it lacks the same interface that allows the user to model objects and easily assign attributes. (modify, to indicate the said objectives)
4. Both the VTK and VRML browsers lack proper spatial analysis tools to manipulate the generated 3-D scenes (i.e. in comparison with 2-D systems).
5. Distributed computing Environment: This design basically replaces the CGI-BIN concept with a JSP page (compiled as a Java Servlet). This architecture works well for many applications, but it does not scale for a large number of simultaneous Web-based clients accessing scarce enterprise resources, since each must establish or share a connection to the content resource in question. For example, if the JSP page accesses a database, it may generate many connections to the database, which can affect the database performance.

References

ActiveWorlds. 1999. Home of 3D chat, Virtual Reality Building Platform. Http://www.activeworlds.com

Agin G. J. and Binford T. O. 1976. Computer descriptions of Curved Objects, IEEE Trans. Comput. C-25, pp439-449.

Agoston M. K. 1976. Algebraic Topology. Marcel Dekker. New York.

Ala S.R. 1991. Design Methodology of Boundary Data Structures. ACM 089791-4279/91/006/0013.

Arya S., Mount D.M., Netanyahu N.S., Silverman R. and Wu A. Y. 1999. An Optimal Algorithm for Approximate Nearest Neighbour Searching in Fixed Dimensions. ACM Vol. 45, No. 6, 004-5411/99/1100-0891

Atkinson M., Bancilhon F., DeWitt D., Dittrich K., Maier D., Zdonik S.1989: The object-oriented database system Manifesto. In Kim W., Nicolas J., Nishio S. (eds): Proceedings of the 1st International Conference on Deductive and Object-Oriented Databases (DOOD'89), Kyoto, Japan, 1989.

Bakman A. 1995. How to Deliver Client/Server Applications that work. Manning Publications. America.

Barrow H. G., Tenenbaum J. M., Bolles R. C., and Wolf H. C. 1977. Parametric correspondence and chamfer matching: Two new techniques for image matching, in Proceedings of 5th International Joint Conference on Artificial Intelligens, Cambridge, MA, USA, August 1977, pp. 659-663.

Batty M., Dodge M., Doyle S. and Smith A. (1998). Visualising Virtual Urban Environments, CASA Working Paper1. Center for Advanced Analysis, University College London

Baumgart B. G. 1974. Geometric modeling for Computer Vision, PhD Thesis, Computer Science Dept. CS-463, Stanford Artificial intelligence Laboratory, Stanford university, California.

Baumgart B. G. 1975. A Polyhedron representation for computer vision, AFIPS National Computer conference,pp 589-596.

Bergen S. 1995. The Validity of Computer Generated Graphic Images of Forest Landscapes, Journal of Environmental Psychology, Vol. 15: pp. 135-146.

Bergen, S.1993. Mitigating Potential Impact to Visual Quality During the Design of Forest Operational Plans. MSc Thesis. University of Washington.

Bernardini, F and Rushmeier, HE: 2000. 3D Model Acquisition, in Eurographics 2000,State of the Art Reports Proceedings, Eurographics Association, Interlaken, August 24 - 25, pp. 41 - 62.

Bhunu S.T. 1999. Asset Management Information System, Water Infrastructure in Harare, Msc Thesis, ITC, The Netherlands.

Blanchin R. and Chiles J. 1991.Geostatistical modelling of geological layers and optimisation of survey design for the channel tunnel, in Pflug and Harbaugh (1991), pp. 251-256

Blinn J.F. 1982.A generalization of Algebraic surface drawing.ACM TOG,1(3), July 1982.

Boar B.H. 1993. Implementing Client/Server Computing, A strategic approach. New York: McGraw-Hill.

Borgerfors G. 1995.On Digital Distance Transforms in Three Dimensions. Computer Vision, Graphics and Image Understanding. Vol.64 .No. 3. November. Pp 368-376. Article No. 0065

Borgerfors G. 1986.Distance Transformations in Digital Images. Computer Vision, Graphics and Image Processing 43. pp344-371.

Borgerfors G. 1984. Distance Transformations in Arbitrary Dimensions. Computer Vision, Graphics and Image Processing 27, pp321-345

Bosselmann P. 1998. Representation of Places: Reality and Realism in City Design. University of California Press. ISBN 0-520-20658-4.

Boyse J.W. 1979. Interference Detection Among Soilds and Surfaces, Commun. ACM22,1.pp3-9.

Bradley A. Payne and Aurthur W. Toga. 1992. Distance Field Manipulation of Surface Models, IEEE, 1992. UCLA School of Medicine.

Breen D. E. and Mauch S.1999. Generating shaded offset surfaces with distance, closest-point and colour volumes. In International Workshop on Volume Graphics, pages 307-320, Swansea, United Kingdom, 1999.

Breen D. E., Mauch S., and Whitaker R. T. 1998. 3D scan conversion of CSG models into distance volume. In IEEE Symposium on Volume Visualization, pages 7-14, 1998.

Bric V., Pilouk M. and Tempfli K.(1994). Towards 3D GIS; Experimenting with Vector data Structure. ISPRS Commission IV, 1994, Vol. 30, Athens.

Brisson E. 1989:Representing geometric structures in **d** dimensions: topology and order, in Proceedings of the 5th annual symposium on Computational Geometry, Saarbruchen, pp218-217.

Brodlie K. Dykes J., Gillings M., Haklay M. E., Kitchin R. and Kraak M. J. 2002. Geography in VR context, Virtual reality in Geography, Published by Taylor and Francis, London and New York.

Brunet P., Ayala D., Juan R. and Navazo I. 1985. Object representation by means of non-minimal division quadtrees and octrees ,ACM Transactions on Graphics, vol 4,1,pp 41-59, 1985.

Brown M.I., Kinder D.B. and Chen W. 2002. Multi-resolution virtual environments as a visualisation frontend. Virtual Reality in Geography ed. Fisgher P. and Unwin D. Taylor and Fracis Inc. New York.

Buehler K., McKee L. (eds)1996. The OpenGIS Guide; Introduction to Interoperable Geoprocessing. Open GIS Consortium, Inc., Wayland, MA, 1996.

Burdea G. and Coiffet P. 1993. Virtual Reality, Edition Hermes, Paris.
Canada, October 4-8.

Carey, R., and Bell, G., 1997, T he Annotated VRML 2.0 Reference Manual (Reading, MA:Addison-Wesley Longman).

Carter H. 1981. The Study of Urban Geography. London: Edward Anorld.

Chen M.J., Gursoz E. and Prinz F.B. 1993. Integration of Parametric Geometry and Non-Manifold Topology in Geometric Modelling. ACM.**Chen S.** 1991. Computer-aided Visual Simulation in an Urban Context. MSc Thesis. College of Environmental Science and Forestry, State University of New York, NY.

Chorafas D.N.1989. Systems architecture and systems design. New York:McGraw-Hill

Coors V. 2003. 3-D GIS in Networking Environment, Computer , Enviroment and Urban Systems, Vol 27(4), pp345-357.

Cuisenaire O. and Macq B. 1999. Fast Euclidean distance transformation by propagation using multiple neighborhoods, Computer Vision and Image Understanding, v.76 n.2, p.163-172, Nov. 1999

Danielson P. E.1980. Euclidean distance mapping, Computer Vision, Graphics, Image Processing, vol. 14, pp. 227-248, 1980.

De Floriani, L and Puppo, E and Cignoni, P and Scopigno, R: 1999. Level-of-Detail in Surface and Volume Modeling, Tutorial in Eurographics '99 Proceedings, Eurographics Association, Milan, september.

Debevec P., Taylor C., Malik J.1996: Modeling and Rendering Architecture from Photographs. In Siggraph 96 Computer Graphics Proceedings, New Orleans, LA, 1996.

Dodge M. Smith A. and Doyle S.1997. Urban Science. GIS Europe,6,10,26-9.

Dodge M., Doyle S., Smith A. and Fleetwood S. (1998). Towards The Virtual City: VR & Internet GIS for Urban Planning. Virtual Reality and Geographical Information Systems, Birkbeck College. London.

Dong F. 1991. (Unpublished).Three-Dimensional models and applications in subsurface modeling, M.Sc. Thesis, University of Calgary, Canada.

Eastman C. M. and Weiler K. 1979. Geometric Modelling using the Euler Operators. Res.Rep.78, Institute of Physical Planning, Carnegie-Mellon University, Pittsburgh, Pa.

Eastman C.M. 1982. Introduction to computer aided design Course Notes Carnegie Mellon University, Pittsburgh, PA, USA.

Eck M. and Hoppe H. 1996. Automatic reconstruction of B-spline surfaces of arbitrary topological type. International Conference on Computer Graphics and Interactive Techniques. Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. ISBN:0-89791-746-4 . ACM Press New York, NY, USA

Edwards J. and DeVoe D.1997. 3-Tier Client/Server At work. Wiley Computer Publishing. New York.

Engenhofer M. J. and J. R. Herring.(1992). Categoring topological relations between : Proceedings of Fourth International Symposium on SDH, Zurich, Switzerland,pp 803-813.

Elmqvist N. 2001 (unpublished). 3Dwm: Three-Dimensional User Interfaces using fast constructive solid geometry. MSc thesis. Chalmers University of Technology, Department of Computing Science, Sweden.

Fairbairn, D., and Parsley, S., 1997, The use of VRML for cartographic presentation.Computers & Geosciences, 23, 475–481.

Farin G. 1997. Curves and Surfaces for Computer Aided Geometric Design. A Practical Guide. Academic Press. San Diego.

Foley J. D., van Dam A., Feiner S. K. and Hughes J. F. 1991. Computer Graphics, Principles and practice. Addison-Wesley Publishing Company, Reading, Massachusetts, California, USA.

Foley J.D. and van Dam A. 1983. Fundamentals of Interactive Computer Graphics.

Foley, J. and A. van Dam. 1996. Computer Graphics: Principles and Practices. Second edition. Addison-Wesley Publishing Company, Inc.

Frank A. and Buyong T. 1989."Geometry for Three-Dimensional GIS in Geoscientific Applications", in Turner (1989)

Frisken S.F. and Perry R.N. 2001. Kizamu: A System For Sculpting Digital Characters. SIGGRAPH 2001 conference proceedings, MERL Mitsubishi Electric Research Laboratories, Cambridge, Massachusetts.

Frisken S.F., Perry R.N., Rockwood A.P., and Jones T.R. 2000. "Adaptively Sampled Distance Fields: A general representation of shape for computer graphics", Computer Graphics (proc. SIGGRAPH '00), vol. 34, no.4, pp.249-254, July 2000.

Gibson S.F. 1998. "Using Distance maps for smooth surface representation in sampled volumes: in IEEE Symposium on Volume Visualisation, Los Alamitos California, Oct. 1998, IEEE, pp.22-30, IEEE Computer Science Society Press.

Gray J. and Reuter A. 1993. Transaction Processing: Concepts and Techniques. San Mateo, CA: Morgan Kaufmann.

Guibas L. and Stolfi J. 1985. Primitives for Manipulation of general subdivisions and the compuatation of Voronoi diagrams, ACM Transactions on Graphics, 4, 2. pp 74-123.

Gursoz El. L., Choi Y. and Prinz F. B. 1990. Vertex-Based representation of non-manifold boundaries, Geometric Modelling for Product Engineering (M.J. Wozny, J.U. Turner and K. preiss,eds), New York: North-Holland, pp107-130.

Hall M. 2003. Core Servlets and JavaServerPages. Online Book version: Http://www.javalobby.org

Hawryszkiewycz I.T. 1994, Introduction to Systems Analysis and Design, 3rd edn, Prentice-Hall, Australia. ISBN: 0-13-148404-4

Hedley N. R., Billinghurst M., Postner L. and May R. n.d.
Human Interface Technology Laboratory. Explorations in the use of Augmented Reality for Geographic Visualization. Human Interface Technology Laboratory University of Washington

Hilary M.H. and David J. U. 1994. Visualization in Geographical Information Systems. Wiley publishers.

Holzner S. 2002. Teach Yourself JavaServer Pages in 21 days. SAMS. USA.

Howe D.R. 1989. Data Analysis For Data Base Design. J. W. Arrowsmith Ltd, Bristol, UK.

Isdale, J., 1998. What is Virtual Reality? A Web-Based Introduction, version 4, draft 1. http://www.isx.com/~jisdale/

Jones M.W. and S. Richard. 2001. Hybrid distance field computation for volumetric objects.In Arie Kaufman, Bill Lorensen, and Klaus Mueller, editors, Proceedings International Workshop on Volume Graphics *2001*, pages 121-133, NY, USA, 2001. SUNY at Stony Brook.

Jones W and Chen M. 1994. A new approach to construction of surfaces from countour data, Computer Graphics forum 13(3) 75-84 [ISSN 0167-7055]

Kauffman J. ; Spencer K.and Birmingham T. W :1999. Beginning ASP databases Wrox Press.

Kavorous M., Reeler E.C., Masry S.E. and J.R.Smart. 1988. An Advanced Geo-Information System for Mining Applications, Computer Applications in the Mineral Industry, Fytas, Collin and Singhal (Eds.) A. Balkema, Rotterdam, 1988, pp. 511-515

Kettner L. 1998. Designing a Data structure for Polyhedral Surfaces. SCG , ACM 1998.Mineappolis Minnesota USA.

Kettner L. 1999. Using Generic Programming for Designing a data structure for Polyhedral Surfaces. Computational Geomatry, Theory and Applications 13, 65-90.

Langford M.L. and Unwin D.J. 1994. Generating and mapping population surfaces within a geographical information system.Cartographic journal, 31,1,21-6.

Laurini R. and Thompson D. 1996.Fundamentals of Spatial Information Systems. Academic Press, Harcourt and Company publishers, London.

Laurini R. and Thompson D.1992. Fundamentals of Spatial Information Systems. Academic Press, 1992.

Leclerc Y.: TerraVision II. 1997. Using VRML to Browse the World. In Data Visualization '97 Proceedings http://www.datavis97.geoplace.com/Proceedings/DV970000proc.html), St. Louis, Missouri, 1997.

Liarokapis F., Sylaiou S., Basu A., Mourkoussis N., White M. and Lister P.F.. 2004. An Interactive Visualisation Interface for Virtual Museums. The 5th International Symposium on Virtual Reality, Archaeology and Cultural Heritage. Cain K., Chrysanthou Y., Niccolucci F. and Silberman N. (Editors) . pp1-10.

Liggett R., Friedman S., Jepson W.1995. Interactive Design/Decision Making in a Virtual Urban World: Visual Simulation and GIS. In Arcview User Conference Proceedings, 1995.

Liggett, R. and W. Jepson, 1995. Implementing an Integrated Environment for Urban Simulation: CAD, Visualization, and GIS in Visual Databases in Architecture (Koutamanis, A., H. Timmermans, I. Vermeulen, Eds.), pp. 145-160. Avebury, Ashgate Publishing Limited, England.

Liggett, R., S. Friedman and W. Jepson, 1998. Interactive Design/Decision Making in a Virtual Urban World: Visual Simulation and GIS. Internal Report, School of Architecture, University of California at Los Angeles, CA.

Liggett, R.S. and W.H. Jepson, 1995. An Integrated Environment for Urban Simulation, in Environment and Planning B: Planning and Design, Vol. 22, pp. 291-302.

Linthicum D.S. 1997. Client/Server and Intranet Development, Wiley Computing Publishing.

MacEachren, A. M., and Kraak, M. J., and Verbree, E., 1999, Cartographic issues in the design and application of geospatial virtual environments. In Proceedings of the 19th International Cartographic Conference 14–21 August, Ottawa, Canada.

Mäntylä M. 1988 .An Introduction to Solid Modeling. Principles of Computer Science. Computer Science Press, Maryland, U.S.A

Martin J. and Leben J. 1995. Client/Server Databases, Enterprise Computing. Prentice Hall, New Jersey.

Martin, D., & Higgs, G. 1997. The visualization of socio-economic GIS data using virtual reality tools. Transactions in GIS, 1(4), 255±265.

Molenaar M. 1992. A topology for 3D Vector Maps, ITC Journal, 1992-1,25-33.

Molenaar M. 1998. An introduction to the theory of spatial object modelling, Taylor&Francis, London. Proceedings of the Earth Observation & Geo-Spatial

Web and Internet Workshop '98 . Instituts für Geographie der Universität Salzburg. ISBN: 3-85283-014-1

Müller H. 1998. Experiences with Semiautomatic Building Extraction. In Third Course in Digital Photogrammetry, Bonn, Germany. Institute for Photogrammetry at Bonn University and Landesvermessungsamt Nordrhein-Westfalen.

Mullikin, J.C. 1992. "The vector distance transform in two and three dimensions." CVGIP: Graphical Models and Image Processing 54(6): 526-535.

Naylor B. n.d. Computational Representations of Geometry.Spatial Labs, USA (Unpublished).

Naylor B., Amanatides J. and Thibult W. 1990. Merging BSP trees Yields Polyhedral Set Operations, ACM Computer Graphics SIGGRAPH '90, 24(4): 115-162.

Peng, Z., 1999, An assessment framework for the development of Internet GIS. Environment and Planning B: Planning and Design, **26**, 117–132.

Pfund M. and A. Carosio 1999.Automatic generation and management of topologically structured 3-D objects . Proceedings of 'International Workshop on Dynamic and Multi-Dimensional GIS, Beijing, 4-6 October 1999.

Pilouk M and Tempfli K. 1999. A tetrahedron-based 3D vector Data model for Geo-Information, Advanced Geographic Data Modelling, New series, Number 40,1999.

Plewe, B. 1997. GIS online: Information, retrieval, mapping and the Internet. Santa Fe, NM: OnWord Press.

Preiss K. 1979. A procedure for checking the topological consistency of 2-D or 3-D finite element mesh. Prceedings 16th Design Automation conference. San Diego, Calif. pp200-206.

Preparata F.P. and Shamos M. I.1985. Computational Geometry: an Introduction, Springer Verlag. Proceedings of ISPRS, Vol. XXXI, Part B4, Vienna, Austria, pp859-867.

Ranziger, M., & Gleixner, G. 1997. GIS-datasets for 3D urban planning. In S. Hodgson, M. Rumor, & J. J. Harts (Eds.), Proceedings Third JEC-EGI Vienna (pp. 298-307). Netherlands: 105 Press.

Ranzinger M., Gleixner G.1995. Changing the city: datasets and applications for 3D urban planning. In GIS Europe, 3/1995, pp. 28-30.

Raper J. and B. Kelk . 1991. Three-Dimensional GIS in: Geographical Information Systems: Principles and Applications D.J. Maguire, Goodchild M. and Rhind D (editors), Longman GeoInformation, pp 219-317.

Raper, J. 1989 Three-Dimensional Applications in Geographical Information Systems, Taylor & Francis, Hampshire, U.K.

Raper, J. 1993. Interfacing GIS with Virtual Reality technology, in Association for Geographic Information conference Proceedings, Birmingham, November 16-18,

Reddy D. R. and Rubin S. 1978. Representation of Three Dimensional Objects. Rep.CMU-CS_78-113, Dep Computer Science, Carnegie-Mellon Univ.,Pittsburgh, Pa., April 1978.

Requicha A. A. G . 1977. Mathematical models of rigid solid objects. Tech.memo.28, Production Automation Project.University Rochester, Rochester, N.Y., USA

Requicha A. A. G. and Voelcker H. B. 1985. Boolean Operations in Solid Modelling: Boundary Evaluation and Merging Algorithms, Proceedings of IEEE, Vol. 73, No. 1.

Requicha A. A. G..1990. "Representations for Rigid Solids: Theory, Methods and Systems", Computing Surveys, Vol.12, No.4 , December 1980. pp 437-463

Rhyne T.M. 1997. Going Virtual with Geograpphic Information nd Scientific Visualization. Computers and GeoSciences, 23(4): 489-491.

Rohlf J. and Helman J.1994. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. In Siggraph 94 Computer Graphics Proceedings, pp. 381-395, 1994.

Rosenfeld A. and Pfaltz J. L. 1966. Sequential Operations in Digital Picture Processing, In:J. Assoc. Comput. Mach. 12 (4), 471-494.

Rossignac J. and Banerjee R.1996. Topologically exact evaluation of polyhedra defined in CSG with loose primitives", Computer Graphics Forum, Vol. 15, No. 4, pp. 205-217, 1996

Rossignac J. and O'connor M. 1989. A dimension-independent model for point sets with internal structures and incomplete boundaries, in Geometric modeling for Product Engineering, Eds.M Wosny, J. Turner, K Preiss, North-Holland, pp145-180.

Rossignac J. R. 1986. Constraints in Constructive Solid Geometry.Design Automation Sciencve Project, Manufacturing Research Dept., IBM Thomas J. Waston Research Centre. Interactive 3D Graphics Proceedings. New York.

Rossignac J., Borrel P.1992. Multi Resolution 3D Approximations for Rendering Complex Scenes. In Computer Science, IBM T.J. Watson Reseach Center, New York, 1992.

Rossignac J. and Requicha.1984. A. Constant-Radius Blending in Solid Modeling,. ASME Computers In Mechanical Engineering (CIME), Vol. 3, pp. 65-73, 1984.

Rozendaal R.(2000). Interactive visualization using 3D Graphics –an Archaeological Case Study. Unpublished Master of Science Thesis. Geomatics Dept. University of Cape Town. South Africa.

Samet H. 1990. Applications of Spatial Data Structures, Reading, MA, Addison-Wesley.

Samet H. 1994. The Design and Analysis of Spatial Data Structures. University of Maryland, Addison-Wesley Publishing Company, Inc. Reading, Massachusetts. USA.

Samet H.1984. The quadtree and related hierarchical structures, ACM Computing Surveys 16, pp. 187-206, 1984.

Satherley R. and Jones M.W. n.d. Vector-City Distance Transform. (unpublished). Department of Computer Science, University of Wales, Swansea, Singleton Park, Swansae, UK.

Schroeder W, Lorensen W., and Lintchcum S. 1994. Implicit modelling of swept surfaces and volumes. In proceedings of Visualisation '94, pp. 40-45.IEEE.

Scriven M. 1991. Evaluation Thesaurus. Newbury Park, CA:Sage Publications

Scriven M. 2000. Evaluation Core Notes. Claremont graduate University. Available Http://eval.cgu.edu/lectures/lecturen.htm

Shi W.Z. and Guo W. 2002. Topological Relations between spatial objects with uncertainity. In: Shi, W.Z., Fisher P.F., Goodchild M.F.(Eds), Spatial Data Quality. Taylor and Francs, London, pp 50-61.

Singhal S. and Zyda M. 1999. Networked Virtual Environments. Addison-Wesley,1999.

Stouffs R., Krishnamurti, Eastman C.M and Assal H. (n.d.). Non-"Standard" Solid Representations. Department of Architecture, Carnegie Mellon University, Pittsburgh and School of Architecture and Urban Planning, University of California, Los Angeles.

Streilein, A., 1996. Utilization of CAD Models for the Object Oriented Measurement of Industrial and Architectural Objects in International Archives of Photogrammetry and Remote Sensing, Vol. XXI: No. B5. pp. 548-553.

Suter M., Nüesch D.1995. Automated Generation of Visual Simulation Databases Using Remote Sensing and GIS. IEEE Visualization '95, pp. 86-93, Atlanta, GA, 1995.

Suter M., Nüesch D.1995.Automated Generation of Visual Simulation Databases Using Remote Sensing and GIS. IEEE Visualization '95, pp. 86-93, Atlanta, GA, 1995.

Tilove R. B. 1980. Set6 memebership classification : A unified approach to geometric intersection problems. IEEE Trans. On Computers, Vol. C-29.No. 10, pp 874-883.

Tokumasu S., Kunitomo Y., Ohta Y., Yamamoto S. and Kakajima N. 1983. Solid Model in Geometric Modelling Systems: HICAD. Annual ACM IEEE Design Automation Conference Proceedings of the 20th conference on Design automation. ISBN:0-8186-0026-8 IEEE Press Piscataway, NJ, USA

United Nations. 2003. Report: The Environmental Information portal: Urban population as a percentage of total population. http://earthtrends.wri.org/text/POP/variables/448.htm

Vacca, J., 1998. VRML clearly explained, Academic Press Professional.

Van Haecke B. 1997. JDBC: Java Database Connectivity. IDG Books Worldwide Inc. USA.

Voelcker H.B., Middleditch A.E., Zuckerman P.R., Fisher W.B., Nelson T.S., Requicha A.A.G. and Shopiro J.E. 1974. Discrete Part Manufacturing, Theory and Practice. Part 1. Tech 1. Production Automation Project. University Rochester, Rochester, New York.

Watt A., Watt M.1992. Advanced Animation and Rendering Techniques. Addison-Wesley Publishing 1992.

Watt A.1993. 3D Computer Graphics, Addison-Wesley Co., England, 500p.

Weibel, R. and Heller, M. (1991). Digital Terrain Modeling. In: Maguire, D.J., Goodchild, M.F. and Rhind, D.W. (Hrsg.). *Geographical Information Systems: Principles and Applications*. London: Longman, 269-297.

Weiler K. J. 1985. Edge-based data structures for solid modeling in curved surface environments IEEE computer graphics and applications, Vol. 5, No. 1, pp21-40.

Woo T.C. 1985. A combinatorial analysis of Boundary data schemata. IEEE CG &A

Worboys M. F. 1994. A Unified Model of Spatial and Temporal Information, Computer Journal, 37(1), pp. 26-34, 1994

Worboys M.F. 1995. GIS A computing perspective. Taylor and Francis Ltd. London.

Yamaguchi F, and Tokieda T. 1985. Bridge edge and triangulation approach in solid modeling in Kunii T L(ed) Frontiers in Computer graphics, Springer Verlag.

Yamaguchi Y., Kobayashi K. and Kimura F. 1990. Geometric Modeling with Generalised Topology and Geometry: IFIP WG5.2 on Geometric Modeling, Rensselaer ville, New York.

Zlatanova S. 2000. 3D GIS For Urban Development. ITC. Enschede. The Netherlands.(Dissertation).ISBN90-6164-178-0

Zlatanova, S and Gruber, M: 1998. 3D GIS on the Web in proceedings of ISPRS, Com. IV, Stuttgart, 7-10 September pp. 691-699.

Zlatanova, S: 1999. VRML for 3D GIS in Proceedings of the 15th Spring Conference on Computer Graphics, 28 April-1May, Budmerice, Slovakia, pp. 74-82.

Appendix A:
Logical designs of representations

This section presents the logical designs of analysed b-reps as well as the B-rep component of the H3-D.

The E-R modelling notation includes:

- Underlining with a solid line to all key identifiers or unique identifiers
- Definition of enterprise or modelling rules between two entities in order to normalise the relationships (see Howe, 1989).
- **May** implies non-obligatory, and **Must** implies obligatory.

H3-D database modelling

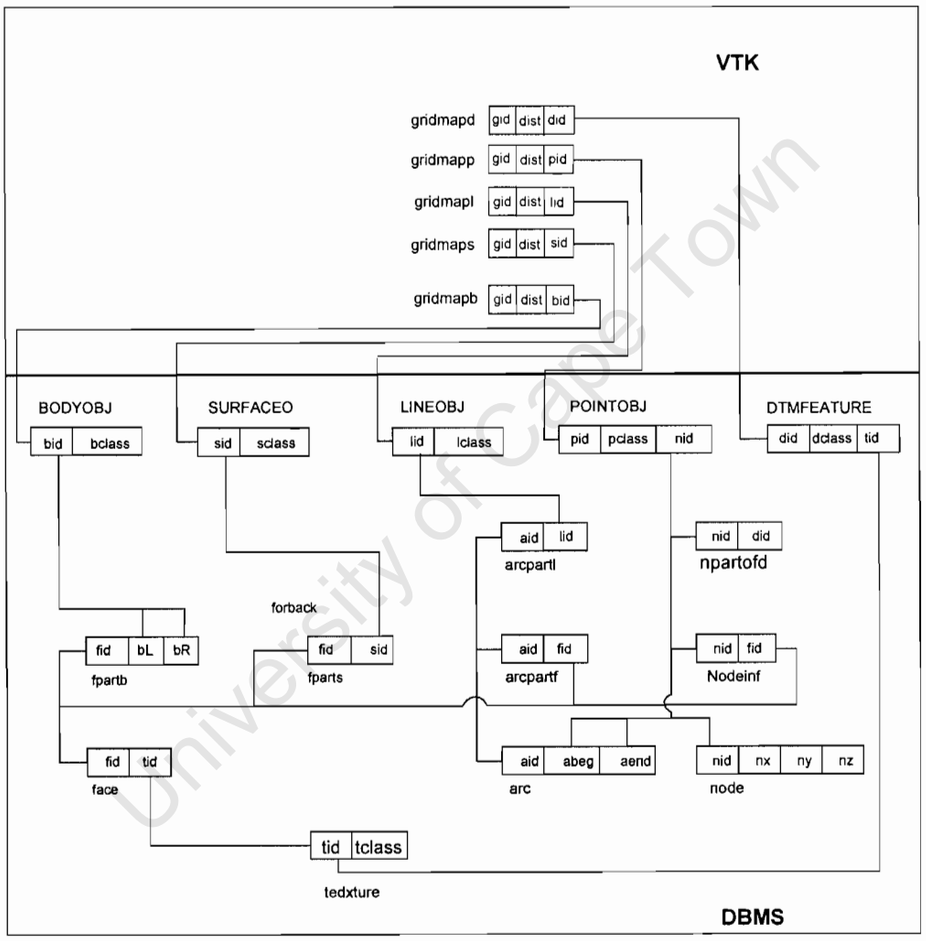


Figure A.1. H3-D Logical design (as implemented)

The skeleton tables given below, provide an overview of the data that is maintained within the Database management system. Figure A.1, illustrates the implementation that was done within the DBMS. It has to be noted that the DF generates it's scene from this data stored using the necessary filters (algorithms) as stated in appendix D and Chapter 6.

The skeletons Tables
Node table: Node (*nid,nx,ny,nz*)
Arc table: Arc (*aid*,aclass)
Face table: Face (*fid,fclass,tid*)
Surface table: Surface (*sid,sclass*)
Body table: (*bid,bclass*)
Line table: (*lid,lclass*)
Point table: (pid, pclass, *nid*)
DTM table:(*did,dclass*)
Texture table: (*tid*,tclass)

Enterprise rules:

1. Node-arc relationship
   An arc must have a start node and an end node.
   A node may be part of at least one arc.
   Table: Node/Arc (*nid, arcbeg,arcend)*

2. Node/ Face relationship
   This relationship models the singularity node contained within a face.

   A node may be contained by a face
   A face may contain one node or more.
   Table: Node/Face(*nid,fid*)

3. Arc-Line relationship

   An arc may be part of at most one line
   A line must be constituted by at least one arc.
   Table: Arc/Line (*aid,lid)*

4. Arc-Face Relationship:
   An arc may be part of at least one face
   A face must be constituted by at least three arcs.
   Table: Face/Arc(*fid,aid*)

5. Face-surface relationship:
   A face may be part of at most one surface.
   A surface must be created from at least one face.
   Table: Face/Surface (*fid,sid)*

6. Face –Body relationship:
   A face may be part of at least one body.
   A body must be constituted by at least four faces.
   Table: Face/Body (*fid,bid)*

7. Texture-Face relationship:
   A face must be have one texture type draped on it
   A texture type may be applied on at least one face
   Table: Texture (*fid,tid)*

8. Texture-DTM relationship:
   A texture type may be draped onto at least one DTM surface.
   A DTM surface must have at least one texture type
   Table: Texture/DTM (*did,tid*)

10. Node-DTM relationship:
    A node may be used in the implementation of a DTM
    A DTM must have at least one node in its generation
    Table: node/DTM (*did,nid*)

UDM database modelling (see Coors, 2003):
Node(*Node-id,……..)*
Point(*Point-id, Node-id,……*)
Face(*Face-id, Body-Right, Body-Left,…..*)
Face/Node(*Face-id, Node-id,……..*)
Line(*Line-id, ,….*)
Node/Line(Line-id, Node-id,….)
Surface(*Surface-id, ,…..*)
Surface/Face(Surface-id,Face-id,…)
Body(*Body-id, Face-id,…….*)

Note: for each primitive, a base geometry attribute is stored.

3D FDS database modelling (see Zlatanova, 2000):

skeleton tables for 3D FDS are given below:

Node(*nid,xc,yc,zc*)

BodyObject(*bid, bodyclass*)

PointObject (*pid, pclass, nid)*

Arc(*arcid, arcbeg, arcend*)

apartofl(*arcid, apartofl*)

LineObject(*l-id, lclass*)

Edge(*fid,enoseq, arcid, forback*)

Face(*fid, fpartofs, bidleft, bidright,texturef)*

aisonf(*arcid,aisonf)*

Surface(*sid, sclass*)

arcinb(*arcid, aisninb)*

nodeonf*(nid, aisonf)*

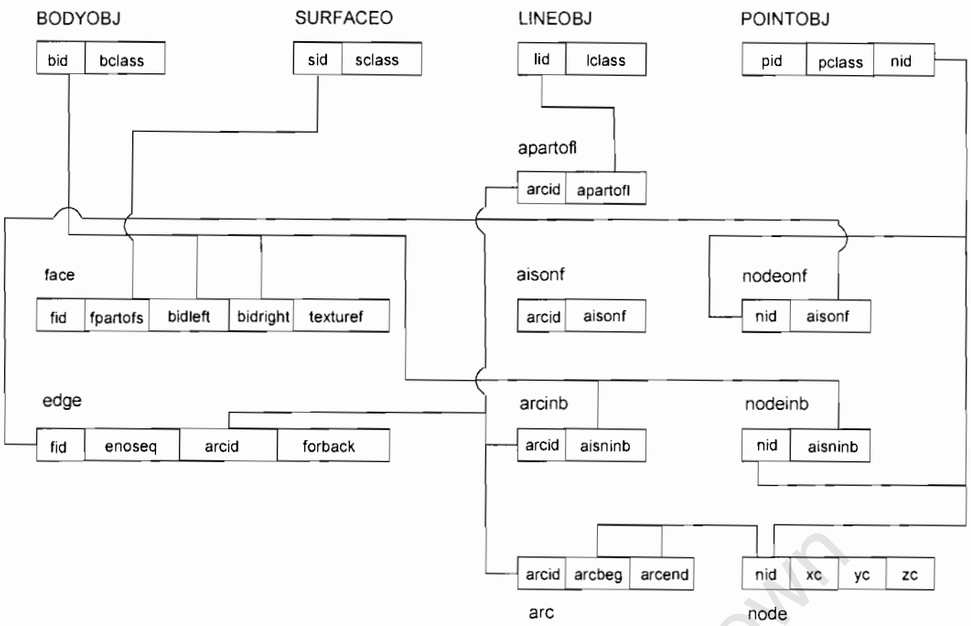


Figure A.2. 3D FDS logical design

Winged-Edge (see table 3.1)database modelling

Edge (*Edge-id, Vertex(Start, End),Face (Left, Right) Left Traverse(Predecessor, Successor), Right Traverse (Predecessor, Successor)*)

Vertex (vertex-id, Incident Edge)
Face(face-id, Incident edge)

SSM database modelling

NODE (*nid,rtree,xc,yc,zc)*
POINT_DA (*pid, nid, rtree, pshape,psize*)
POINT_B(*pid,El,ER*)
POINT_T(*pid, use*)

NINB(*nid,bid*)
LINE_D(*lid,seq,nid*)
LINE_A(*lid,rtree,color,lshape,lsize*)
LINE_B(*lid,El,ER*)
Line_T(*lid,use*)

FACE(*fid,rtree,enoseqf,,nid*)

FINB(*fid,bid)*

SURF_D(*sid,enoseqs,fid*)
SURF_A(*sid, rtree, color, tid*)
SURF_B(*sid,El,ER*)
SURF_T(*sid,use*)

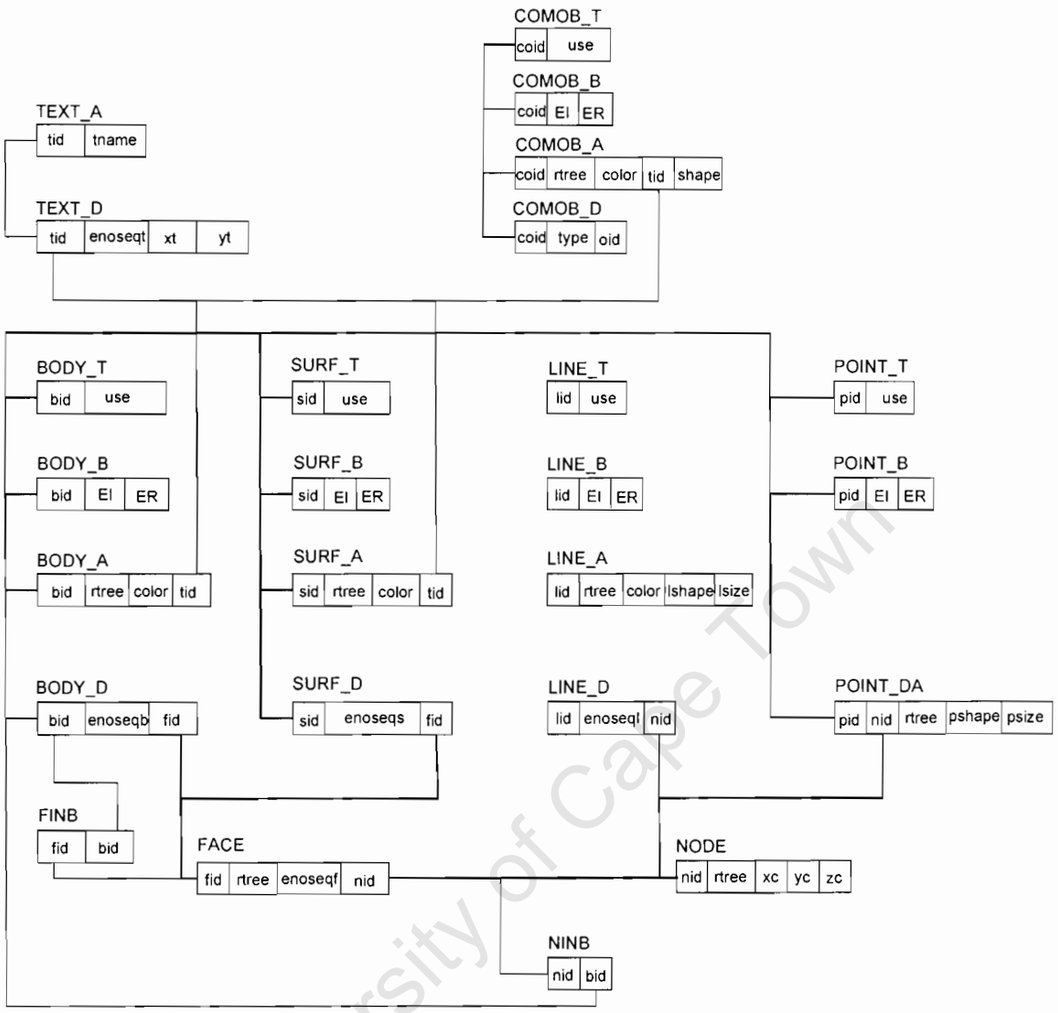


Figure A.3. SSM logical design

BODY_D(*bid,enoseqb,fid*)
BODY_A(*bid,rtree,color,tid*)
BODY_B(*bid,El,ER*)
BODY_T(*bid,use*)

COMOB_D(*coid,type,oid*)
COMOB_A(*coid,rtree,color,tid,shape*)
COMOB_B(*coid,EI,ER*)
COMOB_T(*coid,use*)

TEXT_D(*tid,enoseqb,xt,yt*)
TEXT_A(*tid,tname*)

Appendix B:

Algorithms for the H3-D representation

Algorithms: B1:

The Nearest neighbour (*NN*)search is defined as :

Given a data set $\boldsymbol{S}$ containing $\boldsymbol{N}$ $\boldsymbol{n}$-dimensional points $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots\ldots\ldots \boldsymbol{p}_N$ and a query point $\boldsymbol{q}$, find the *NN* point from the data set $\boldsymbol{S}$ that is closer to $\boldsymbol{q}$ than an other points in the data set, i.e.

$$\boldsymbol{NN}(\boldsymbol{q}) = \{\boldsymbol{a} \in \boldsymbol{S} \,|\, \forall \boldsymbol{p} \in \boldsymbol{S} : \|\boldsymbol{a} - \boldsymbol{q}\| \leq \|\boldsymbol{p} - \boldsymbol{q}\|\}$$

The search process starts by investigating the cell (node) which point $\boldsymbol{q}$ is located in. The algorithm for this is given as:

First to be considered is the recursive top-down traversal of the index to locate a leaf node containing the query object $\boldsymbol{q}$. Note that the traversal is initiated with the root node (i.e., the node spanning the whole index space). The 1st level hierarchical nodes are investigated for the existence of the query object $q(x_q, y_q, z_q)$.

The criterion for the investigation throughout the different levels of the hierarchical structure is

$$FINDLEAF(QueryObject\, q, NODE\ N) = \begin{pmatrix} (x_N - r_L/2) \leq x_q \leq (x_N + r_L/2) \\ (y_N - r_L/2) \leq y_q \leq (y_N + r_L/2) \\ (z_N - r_L/2) \leq z_q \leq (z_N + r_L/2) \end{pmatrix}$$

where r_L is the cell level for level L and a NODE N will have co-ordinates x_N, y_N and z_N and the algorithm can be summerised as

```
FINDLEAF (QueryObject, NODE)
If QueryObject is in node NODE then
        If NODE is a leaf node then
                Report node NODE
        Else
        For each CHILD of node NODE do
                FINDLEAF (QueryObject, CHILD)
                enddo
        endif
endif
```

The first task is to extend the algorithm to find the Distance Field Value (DFV_f) for the feature of interest which is closest to $\boldsymbol{q}$. Once a leaf node containing $\boldsymbol{q}$ has been found (from the above algorithm), the DFVs contained in that node are investigated. However, the DFV_f closest to $\boldsymbol{q}$ might reside in another node. Finding that node may in fact require unwinding the recursion to the top and descending again deeper into the tree. Furthermore, once that node is found, it does not aid in finding the next nearest DFV_f .

To resolve this dilemma, the recursion stack of the regular top-down traversal is replaced with a *priority queue*. In addition to using priority queue for nodes, DFV_f s are also put on the queue as leaf nodes are processed. The key used to order the elements on the queue is the distance from $\boldsymbol{q}$

A node is not examined until it reaches the head of the queue. At this time, all nodes and DFV_f s closer to the query object q have been examined. Initially the node spanning the whole index is the sole element in the priority queue. At subsequent steps, the element as the heads of the queue (i.e., the closest element not yet examined) is retrieved, and this is repeated until the queue is emptied. Informally, the process can be visualised as: Starts by locating the leaf nodes containing point q (See Figure B1, below).Next imagine a circle centered at q being expanded from the starting radius of 0: this circle is termed , *search region.*Each time the circle hits the boundary of a node region, the contents of that node are put on the queue, and each time the circle hits a DFV_f ,we have found the DFV_f next nearest to q .Note that when the circle hits a node or DFV_f , we are guaranteed that the node or DFV_f is already in the priority queue, since the node that contains it must have been hit (this is guaranteed by consistency condition).

The incremental nearest algorithm is summarised as:
INCNEAREST(QueryObject $\boldsymbol{q}$,SpatialIndex)

Queue ← *NEWPRIORITYQUE()*
ENQUEUE(Queue,SpatialIndex.RootNode,0)
While not IsEmpty(Queue) do
Element ← *DEQUEUE(Queue)*
If Element is DFV_f *then*
While Element=FIRST(Queue) do

```
                DELETEFIRST(Queue)
        Enddo
        Report Element
Elseif Element is a leaf node then
        For each DFV_f in the leaf node Element do
                If DIST(QueryObject q, DFV_f) ≥ DIST(QueueObject q, Element) then
                        ENQUEUE(Queue, DFV_f, DIST(QueryObject q, DFV_f)
                Endif
Endo
Else/*element is a nonleaf node*/
        For each Child node of the node Element in SpatialIndex do
                ENQUEUE(Queue, Child,DIST(QueryObject q,Child)
        Enddo
Endif
enddo
```

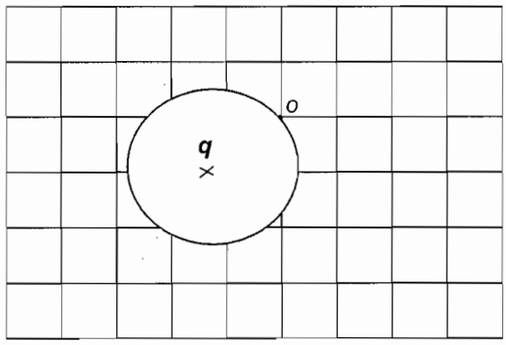


Figure B1.The circle around the query object q depicts the the search region after reporting O as the next nerarest object. For simplicity, the leaf nodes are represented by a grid .Only the shaded leaf nodes are accessed by the incremental nerarest neighbour algorithm. The region with darker shading is where we find objects in the priority queue.

Algorithms B2:Time Complexity Algorithms:

T1: Given vertex V_i, find all the edges linking to it.

T2: Given an arc A_i, find all faces incident on it.

T3: Given a face F_i, find its bounding arcs.

T4: Given a face F_i, find the vertices bounding the face.

T5: Given a face F_i find bodies sharing it.

T6: Given a body B_i, find all the bounding faces.

T7: Given a body B_i, find all the bounding vertices

T8: Given a body B_i, find all the bounding arcs.

T9: Given a line L_i , find all the constituent arcs.

T10: Given a line L_i, find all the constituent vertices.

The following parameters will be adhered to:

Number of edges = N

Number of faces = M

Number of vertices = v

Number of bodies = b

Constant reading time = K

Note: The following geometrical conditions apply:

1. $v \geq N$
2. $N \geq M$
3. $M \geq b$ and $b \geq 0$

In all cases, we assume that the feature under investigation in the relational tables is the bottom record (i.e. worst case scenario). The records search method adopted is the same for all queries and this aids in maintaining consistency in the time complexity comparisons. In some instances, 4 sided faces are considered to allow for easy analysis and this is maintained on all representations.

Spatial Query analysis for the Wing-edged representation:

With the Winged-edge data structure, the queries are mainly directed at the edge table, the vertex table and face table (see Appendix B.). In the winged-edge, the arc is regarded as the edge.

Query *T1:*

The query is spread over the two tables; vertex and edge tables. Firstly, the incident edge is read from the vertex table and the rest of the edges around the vertex are obtained from the edge table. The vertex table is ordered on the *vertex-id* attribute, therefore values in this column are read in constant time K.

The algorithm:

Loop 1: From the Vertex table
When Vertex = V_o *Read Edge* E_o
End Loop 1

Loop 2
 # Search for Edges from the Edge table
when Edge = E_o, Read Start-Edge, End edge

If Start-Vertex=V_o then
Do while LTS.NE.Edge # NE is Not Equal To.
Edge=ED
Read LTS # LTS is Left traverse successor edge
ED=LTS
Print LTS

Else
Read RTS # RTS is Right Traverse Successor
Print RTS # first edge
Edge=RTS
Do while LTS.NE.Edge
Edge=ED
Read LTS # LTS is Left traverse successor edge
ED=LTS
Print LTS

We deduce that the algorithm has time which scales according to:

First table K (constant time for reading)second

Second Table aK where $a < N$, the worst case scenario would be imply that $a \approx N$

Therefore the time complexity can be approximated to NK

Total :T1 complexity is $NK + K$

Therefore the time complexity for this algorithm is Order $O(NK + K)$

Query *T2*:

Since the edge table is ordered on the edges, an edge will be read in constant time K

The algorithm:

For an Edge E_o
*Read edge from **Faces-Left, Faces-Right** fields*
End Loop

We deduce that the algorithm has time which scales according to:

$2K$ (constant) i.e. for reading faces from Left and Right face fields

Total :T2 complexity $2K$

Therefore the time complexity for this algorithm is Order $O(2K)$

Query *T3:*

The query is spread over the two tables; vertex and edge tables. Firstly, the incident edge is read from the vertex table and the rest of the edges around the vertex are obtained from the

edge table. The vertex table is ordered on the *vertex-id* attribute, therefore values in this column are read in constant time K.

The algorithm:

Loop 1: From the Face table
When Face = F_o *Read Edge* E_o
End Loop 1

Loop 2
 # Search for Edges from the Edge table
when Edge = E_o, Read *Face-Left, Face Right*
If Face Right= F_o *then*
Read RTP
Edge=RTP
Do while ED.NE.Edge
De=RTS
Read RTS
Print RTS
Else
Read LTS
Edge=LTS
Do while ED.NE.Edge
Read LTS # LTS is Left traverse successor edge
ED=LTS
Print LTS

We deduce that the algorithm has time which scales according to:

First table K (constant time for reading)

Second Table $4K$ assume a face with 4 sides

Total :T1 complexity is $4K + K$

Therefore the time complexity for this algorithm is Order $O(5K)$

Query *T4*:

This algorithm is similar to T3 in many ways. The difference comes from the fact that for every edge identified, Start and end nodes are read.

The algorithm:

Loop 1: From the Face table
When Face = F_o *Read Edge* E_o
End Loop 1

Loop 2
 # Search for Edges from the Edge table
when Edge = E_o, Read *Face-Left, Face Right*

If Face Right= F_o then
Read RTP
Edge=RTP
Do while ED.NE.Edge
De=RTS
Read RTS
Read Start Vertex, Read End Vertex
Print Start Vertex, Read End Vertex
Else
Read LTS
Edge=LTS
Do while ED.NE.Edge
Read LTS # LTS is Left traverse successor edge
ED=LTS
Read Start Vertex, Read End Vertex
Print Start Vertex, Read End Vertex

We deduce that the algorithm has time which scales according to:

First table K (constant time for reading)

Second Table $2NK$ where N, the worst case scenario would be imply an N sided face and the loops reads edge and vertex values, hence $2K$

Total :T1 complexity is $2NK + K$

Therefore the time complexity for this algorithm is Order $O(2NK + K)$

Spatial Query analysis for the H3-D representation:

Query *T1*:
The analysis is carried out on the arc/node logical table (see Chapter 5).

The algorithm:
Loop 1:Do while $i = 1, v$
Search for V_o *from* ***Edge-Name*** *field*
If Vertex $V_i = V_o$
Read arc from ***Arc-Attribute*** *field*
End Loop 1

We deduce that the algorithm has time which scales according to:

v times + Read time $c_y K$ (constant) where c_y is the worst case scenario, which is the maximum number a vertex will be shared by edges. Assume 6 i.e. in a non-manifold structure for regular shapes.

The time complexity is $v + 6K$

Query *T2*:
The analysis is carried out on the arc/face logical table (see Chapter 5).

The algorithm:

Loop 1:Do while $i = 1, N$
Search for A_o *from* ***Arc-id*** *field*
If Arc $A_i = A_o$
Read face from ***face-Attribute*** *field*
End Loop 1

We deduce that the algorithm has time which scales according to:

N times + Read time $4K$ (constant) assume worst case scenario in urban environments where an arc is shared by four faces (i.e. non-manifolds)

The time complexity is $N + 4K$

Query *T3*:
The analysis is carried out on the arc/face logical table (see Chapter 5).

The algorithm:

Loop 1:Do while $i = 1, M$
Search for F_o *from* ***Face-id*** *field*
If Face $F_i = F_o$
Read arc from ***arc-Attribute*** *field*
End Loop 1

We deduce that the algorithm has time which scales according to:

N times + Read time $4K$ (constant) assume 4-sided faces in order to simplify analysis.

The time complexity scales as $N + 4K$

Query *T4*:
The analysis is carried across two logical tables, Face/Arc and Arc/Node.
This analysis is split into two:

(a) Firstly, the bounding arcs have to be determined (i.e. T3 query):

The time complexity is $N + 4K$

(b) Secondly, vertices for each of the four arcs are determined

The algorithm:

Outer Loop 1:Do while $j = 1,4$
Inner Loop 1:Do while $i = 1, N$

Search for A_j *from* ***Arc-id*** *field*
If Arc $A_i = A_j$
Read vertex from ***node-Attribute*** *field*
End Inner Loop 1
End Outer Loop 1

For each arc, there are two nodes bounding it.
Therefore the time scales according to:

4(N times + Read time $2K$ (constant))

Total for Query T4 = $(N + 4K) + (4N + 8K)$

$= 5N + 12K$

The total complexity for this T4 query scales according to $5N + 12K$

Query *T5*:
The analysis is carried on the face/body logical table.

The algorithm:
Loop 1:Do while $i = 1, M$
Search for F_o *from* ***Face-id*** *field*
If Face $F_i = F_o$
Read body from ***body-Attribute*** *field*
End Loop 1

We deduce that the algorithm has time which scales according to:

M times + Read time $2K$ (constant), the maximum number of bodies per face is 2.

The time complexity scales as $M + 2K$

Query *T6*:
The analysis is carried on the face/body logical table.

The algorithm:
Loop 1:Do while $i = 1, b$
Search for B_o *from* ***Body-id*** *field*
If Body $B_i = B_o$
Read face from ***face-Attribute*** *field*
End Loop 1

We deduce that the algorithm has time which scales according to:

b times + Read time $6K$ (constant), assume a 6-sided tetrahedron, to simplify the analysis.

The time complexity scales as $b + 6K$

Query *T7*:
The analysis is carried out across the Body/Face, Face/Arc and Arc/Node logical tables (see Chapter 5). The investigation is therefore split into three:

(a) Find the bounding faces (Query T6)

The time complexity scales as $b + 6K$

(b) Finding the bounding arcs for each of the 6 faces (6xQuery T3 complexity)

The time complexity scales as $6N + 24K$

(c) For each of the 4 arcs on 6 faces, find the vertices. (6 x Query T4 (b))

The time complexity scales as $6(4N + 8K) = 24N + 48K$

The total time complexity scales as:

$(24N + 48K) + (6N + 24K) + (b + 6K)$

$= 30N + b + 78K$

Query *T8*:
The analysis is carried out across the Body/Face, Face/Arc (see Chapter 5).

The solution to this query is adding up query T7(a) and Query T7(b)

i.e. $(b + 6K) + (6N + 24K) = (6N + b + 30K)$

Query *T9:*
The analysis is carried out across the Line/Arc logical table (see Chapter 5).

Worst case scenario, the number of arcs is equal to the number of lines (i.e. having signle arc lines).

The algorithm:

Loop 1:Do while $i = 1, N$
Search for L_o *from* ***Line-id*** *field*
If Line $L_i = L_o$
Read arc from ***arc-Attribute*** *field*
End Loop 1

Assume worst case of line with c arcs

Therefore The time complexity scales as $N + cK$ (to simplify analysis assume c=4)

Hence the complexity scales as $N + 4K$

Query *T10*:
The analysis is carried out across the Line/Arc and Arc/None logical tables (see Chapter 5).

This is split into two:

(a) Number of arcs (Query T9)

The complexity scales as $N + 4K$

(b) Number of vertices given 4 arcs (Query T4(b)

Therefore the time complexity scales as $4(N + 2K) = 4N + 8K$

The total complexity scales as $(4N + 8K) + (N + 4K)$

$= 5N + 12K$

Spatial Query analysis for the 3D FDS representation:

For a detailed description of the logical 3D FDS logical model, see Molenaar 1990.

Query *T1*:
The analysis is carried out on the arc logical table, and this table is ordered on the arc-id (and not the edge), hence a search is carried out for vertex determination. (see Chapter 5).

Arc(*Arc-id, arc-begin-Node. Arc-end-Node,...*)

The algorithm:

Loop 1:Do while $i = 1, v$
Search for V_o *from* ***Arc-Begin*** *field*
If Vertex $V_i = V_o$
Read arc from ***Arc-Attribute*** *field*
End Loop 1
Loop 2:Do while $i = 1, v$
Search for V_o *from* ***Arc-End*** *field*
If Vertex $V_i = V_o$
Read arc from ***Arc-Attribute*** *field*
End Loop 2

We deduce that the algorithm has time which scales according to:

Loop1: v times + Read time $c_y K$ (constant)

Loop2: v times + Read time $c_y K$ (constant)

where c_y is the worst case scenario, which is the maximum number a vertex will be shared by edges (take c_y = 6, for non-manifold regular shapes)

The time complexity is $2v + 12K$

Query *T2*:
The analysis is carried out on the Edge skeleton table provided below.

Edge *(Face-id, Arc-id, Enoseq, Forward-backward)*

The algorithm:

Loop 1:Do while $i = 1, N$
Search for A_o *from* ***Arc-id*** *field*
If Arc $A_i = A_o$
Read face from ***face-Attribute*** *field*
End Loop 1

We deduce that the algorithm has time which scales according to:

N times + Read time $4K$ (constant) assume worst case scenario in urban environments where an arc is shared by four faces (i.e. non-manifolds)

The time complexity is $N + 4K$

Query *T3:*
The analysis is carried out on the edge skeleton table below::

Edge *(Face-id, Arc-id, Enoseq, Forward-backward)*

The algorithm:

Loop 1:Do while $i = 1, N$
Search for F_o *from* ***Face-id*** *field*
If Face $F_i = F_o$
Read arc from ***arc-Attribute*** *field*
End Loop 1

We deduce that the algorithm has time which scales according to:

N times + Read time $4K$ (constant) assume 4-sided faces in order to simplify analysis.

The time complexity scales as $N + 4K$

Query *T4:*
The analysis is carried on the face table:

(a) Firstly, the bounding arcs have to be determined (i.e. T3 query):

The time complexity is $N + 4K$

(b) Secondly, vertices for each of the four arcs are determined

In this instance, the table ***Arc*** (*Arc-id, Node(arc-begin, arc-end)*) is used.

The algorithm:

Outer Loop 1:Do while $j = 1,4$
Inner Loop 1:Do while $i = 1, N$

Search for A_j *from* ***Arc-id*** *field*
If Arc $A_i = A_j$
Read vertex from ***node-begin Attribute*** *field*
End Inner Loop 1
End Outer Loop 1

Outer Loop 2:Do while $j = 1,4$
Inner Loop 2:Do while $i = 1, N$

Search for A_j *from* ***Arc-id*** *field*
If Arc $A_i = A_j$
Read vertex from ***node-end Attribute*** *field*
End Inner Loop 2
End Outer Loop 2

For each arc, there are two nodes bounding it.
Therefore the time scales according to:

2 x 4(N times + Read time $2K$ (constant))

Total for Query T4 = $(N + 4K) + (8N + 16K)$

$= 9N + 20K$

The total complexity for this T4 query scales according to $9N + 20K$

Query *T5:*
The analysis is carried on the face table. Note that this table is ordered on the Face-id, hence there is constant time reading of values.

Face*(Face-id, Body-left, Body-right, ...)*

The algorithm:

Loop 1:

for F_o *Read body from* ***body-Left, Body-right Attribute*** *fields*
End Loop 1

We deduce that the algorithm has time which scales according to:

Read time $2K$ (constant), the maximum number of bodies per face is 2.

The time complexity scales as order $\boldsymbol{O}(2K)$

Query *T6:*
The analysis is carried on the face table.
Face*(Face-id, Body-left, Body-right,... ...)*

The algorithm:

Loop 1:Do while $i = 1, b$
Search for B_o *from* ***Body-Left*** *field*
If Body $B_i = B_o$
Read face from ***face-Attribute*** *field*
End Loop 1

Loop 2:Do while $i = 1, b$
Search for B_o *from* ***Body-Right*** *field*
If Body $B_i = B_o$
Read face from ***face-Attribute*** *field*
End Loop 2

We deduce that the algorithm has time which scales according to:

Loop 1: b times + Read time $3K$ (constant),

Loop 2: b times + Read time $3K$ (constant),

Assume a 6-sided tetrahedron, to simplify the analysis.

The time complexity scales as $2b + 6K$

Query *T7*:
The analysis is carried out across the Face, edge and Arc tables. The investigation is therefore split into three:

(a) Find the bounding faces (Query T6)

The time complexity scales as $2b + 6K$

(b) Finding the bounding arcs for each of the 6 faces (6xQuery T3 complexity)

The time complexity scales as $6N + 24K$

(c) For each of the 4 arcs on 6 faces, find the vertices. (6 x Query T4 (b))

The time complexity scales as $6(8N + 16K) = 48N + 96K$

The total time complexity scales as:

$(2b + 6K) + (6N + 24K) + (48N + 96K)$

$= 54N + 2b + 126K$

Query *T8*:
The analysis is carried out across the Face and Edge tables.

The solution to this query is adding up query T7(a) and Query T7(b)

i.e. $(2b + 6K) + (6N + 24K) = (6N + 2b + 30K)$

Query *T9:*
The analysis is carried out on the ***Arc-part-of-Line*** table

Arc-part-of-Line *(arc-id, L-id)*

Worst case scenario, the number of arcs is equal to the number of lines (i.e. having simple arc lines).

The algorithm:

Loop 1: Do while $i = 1, N$
Search for L_o *from* ***Line-id*** *field*
If Line $L_i = L_o$
Read arc from ***arc-id Attribute*** *field*
End Loop 1

Assume worst case of line with c arcs

Therefore The time complexity scales as $N + cK$ (to simplify analysis assume c=4)

Hence the complexity scales as $N + 4K$

Query *T10*:
The analysis is carried out across the ***Arc-part-of-Line*** and ***Arc*** tables.

This is split into two:

a) Number of arcs (Query T9)

The complexity scales as $N + 4K$

b) Number of vertices given 4 arcs (Query T4(b)

Therefore the time complexity scales as $8N + 16K$

The total complexity scales as $8N + 16K + (N + 4K)$

$$= 9N + 20K$$

Spatial Query analysis for the SSM representation:

For a detailed description of the SSM logical model, see Zlatanova 2000.

Queries ***T1,T2 ,T3,T8*** and ***T9*** cannot be easily implemented using the SSM representation.

Query *T4*:

The analysis is carried on the face table. He table presents a many-to-many data entry between the faces and the nodes, hence the ordering could be through some other attribute:

Face*(face-id,node-id,....)*

The algorithm:

Loop 1:Do while $i = 1, M$

Search for F_i *from* ***face-id*** *field*
If face $F_i = F_o$
Read vertex from ***node-id Attribute*** *field*
End Loop 1

The complexity for this T4 query scales according to $M + 4K$. Assume a four-sided face

Query *T5:*

The analysis is carried on the ***body*** table. This query is treated like ***T4*** above.

Body *(body-id, face-id,....)*

The algorithm:

Loop 1:Do while $i = 1, M$
Search for F_o *from* ***face-id*** *field*
If Face $F_i = F_o$
Read body from ***body-id Attribute*** *field*
End Loop 1

We deduce that the algorithm has time which scales according to:

M times + Read time $2K$ (constant), the maximum number of bodies per face is 2.

The time complexity scales as $M + 2K$

Query *T6:*
The analysis is carried on the ***body*** table.
Body *(body-id, face-id, ...)*

The algorithm:

Loop 1:Do while $i = 1, b$
Search for B_o *from* ***Body-id*** *field*
If Body $B_t = B_o$
Read face from ***face-id Attribute*** *field*
End Loop 1

We deduce that the algorithm has time which scales according to:

b times + Read time $6K$ (constant),

Assume a 6-sided tetrahedron, to simplify the analysis.

The time complexity scales as $b + 6K$

Query *T7:*
The analysis is carried out across the ***body*** and ***face*** tables The investigation is therefore split into two:

(a) Find the bounding faces (Query T6)

The time complexity scales as $b + 6K$

(b) Finding the bounding vertices for each of the 6 faces (6xQuery T4 complexity)

The time complexity scales as $6(M + 4K) = 6M + 24K$

The total time complexity scales as:
$(b + 6K) + (6M + 24K)$
$= b + 6M + 30K$

Query *T10:*
The analysis is carried out across on the ***Line_D*** table.

Line_D *(Line-id,node-id,....)*

The algorithm:

Loop 1:Do while $i = 1, N$

Search for L_o *from* ***Line-id*** *field*

If Line $L_i = L_o$

Read node from ***node-id Attribute*** *field*

End Loop 1

Assume simple lines which would mean the total number will be N

Therefore the total complexity scales as $N + 2K$

Spatial Query analysis for the UDM representation:

For a detailed description of the logical UDM model, see Coors 2003.

Queries ***T1,T2 ,T3,T8*** and ***T9*** cannot be easily implemented using the SSM representation.

Query *T4:*

The analysis is carried on the face/node table:

Face/node *(face-id ,node-id,....)*

The algorithm:

Loop 1:Do while $i = 1, M$

Search for F_i *from* ***face-id*** *field*

If Face $F_i = F_o$

Read vertex from ***node-id Attribute*** *field*

End Loop 1

The complexity for this T4 query scales according to

$M + 4K$. Assume a four-sided face

Query *T5*:

The analysis is carried on the ***face*** table, note that this table is ordered on the face.

face *(face-id, body(body-left, body- right))*

The algorithm:

Loop 1:

for F_o *from* ***face-id*** *field*

Read body from ***body-left, body-right Attribute*** *fields*

End Loop 1

We deduce that the algorithm has time which scales according to:

Read time $2K$ (constant), the maximum number of bodies per face is 2.

The time complexity scales as $2K$

Query *T6*:
The analysis is carried on the ***face*** table.
face *(face-id, body(body-left, body- right))*

The algorithm:

Loop 1:Do while $i = 1, b$
Search for B_o *from* ***Body-left -id*** *field*
If Body $B_i = B_o$
Read face from ***face-id Attribute*** *field*
End Loop 1
Loop 1:Do while $i = 1, b$
Search for B_o *from* ***Body-right -id*** *field*
If Body $B_i = B_o$
Read face from ***face-id Attribute*** *field*
End Loop 1

It can be deduced that the algorithm has time which scales according to:

2 x (b times + Read time $3K$ (constant))

Assume a 6-sided tetrahedron, to simplify the analysis.

Therefore time complexity scales as $2b + 6K$

Query *T7*:
The analysis is carried out across the ***face*** and ***face/node*** tables The investigation is therefore split into two:

(a) Find the bounding faces (Query T6)

The time complexity scales as $2b + 6K$

(b) Finding the bounding vertices for each of the 6 faces (6xQuery T4 complexity)

The time complexity scales as $6(M + 4K) = 6M + 24K$

The total time complexity scales as:

$$(2b+6K)+(6M+24K)$$

$$=2b+6M+30K$$

Query *T10*:
The analysis is carried out across on the ***Line/Node*** table.

Line/Node *(Line-id,Node-id,....)*

The algorithm:

Loop 1:Do while $i=1,N$
Search for L_o *from* ***Line-id*** *field*
If Line $L_i = L_o$
Read node from ***node-id Attribute*** *field*
End Loop 1

Assume simple lines which would mean the total number will be N

Therefore the total complexity scales as $N+2K$

Appendix C:

Marching Cube Algorithm and 3D spatial operations

1. Matching Cubes

When data is contoured, a boundary is being effectively created between these regions. These boundaries correspond to contour lines (2-D) or iso-surfaces (3-D).

The marching cubes algorithm is a technique for creating 3-D surfaces. The technique is best explained by first introducing the marching squares algorithm, an analogous 2-D technique.

Consider the 2-D distance field grid Figure C.1 , below.

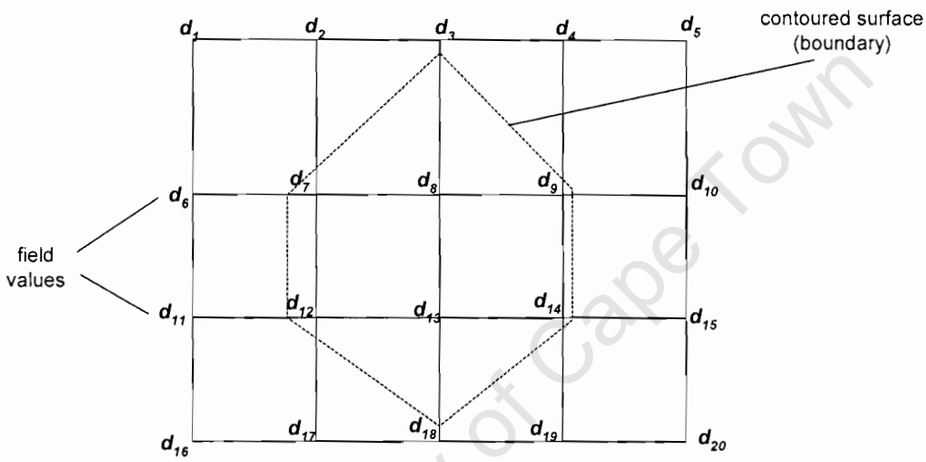


Figure C.1. 2-D contouring

The contour is generated by linear interpolation of the distance field values. Interpolated points are marked on the edges first and then connected to form counters. One approach detects an edge intersection (i.e. the contour passes through an edge) and then tracks it across cell boundaries. Every edge must be checked to see whether other contour lines do exist (in the case of more than one contour line).

Another approach uses the divide and contour technique, treating cells independently. The matching "squares" technique. The basic assumption of this technique (hence the matching cubes

technique), is that a contour (or Iso-surface) can pass though through a cell in a finite number of ways. A case table is constructed that enumerates all possible topological states of a cell given combinations of scalar values at the cell points. A vertex is considered inside the contour if its scalar value is larger than the contour line. Vertices with scalar values less than the contour value are said to be outside the contour. If a cell has four vertices and each vertex can be either inside or outside the contour, there are $2^4 = 16$ possible ways that contour lines pass through the cell. In the case table, attention is not given to where the contour passes through the cell, but whether it passes through the cell. Figure C.2, below shows the 16 combinations.

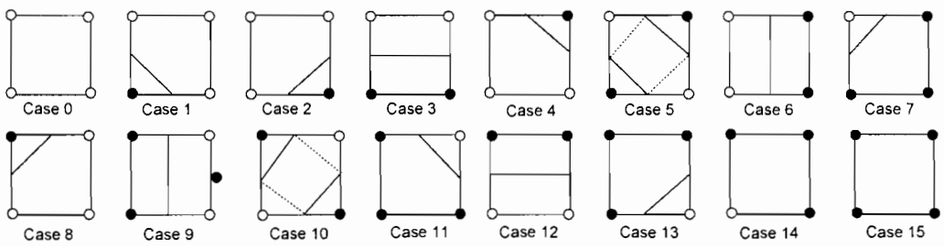


Figure C.2. 16 different marching squares cases. Dark vertices indicate scalar Value is above contour value.

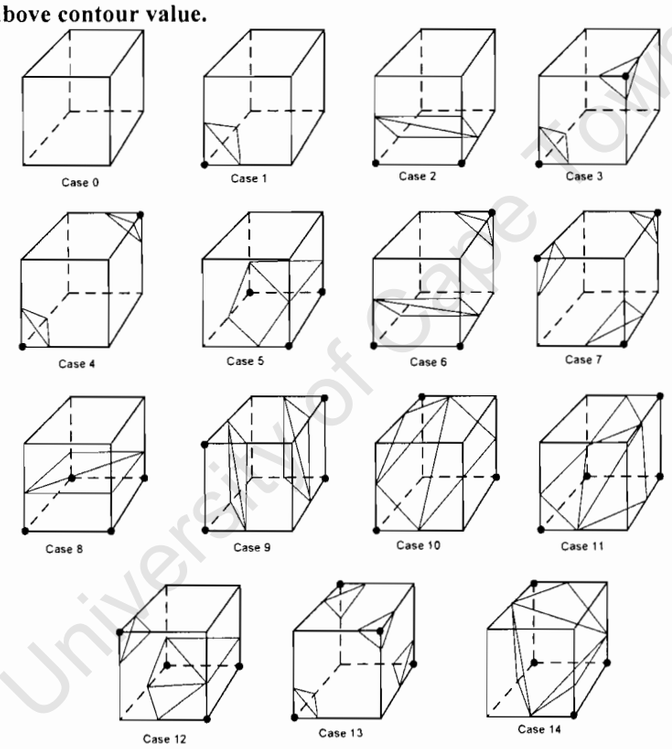


Figure C.3. Marching cubes cases for 3-D iso-surface generation. The 256 possible cases have been reduced to 15 cases using symmetry. Dark vertices are greater than the selected iso-surface value.

The 3-D analogy of matching squares, is matching cubes. There are 256 different combinations of scalar value, given that there are eight points in a cubical cell (i.e. 2^8 combinations). These combinations are reduced to 15

However to simplify the algorithm we can reduce the complexity by taking into account cell combinations that duplicate under the following conditions.

- Rotation by any degree over any of the 3 primary axis
- Mirroring the shape across any of the 3 primary axis
- Inverting the state of all corners and flipping the normals of the relating polygons.

2. Tri-linear interpolation

In three dimensions, there are eight surrounding grid points. To combine their respective influences will require a trilinear interpolation (linear interpolation in each of three dimensions). The interpolation assumes that the gradient between the points is constant (Figure C4).

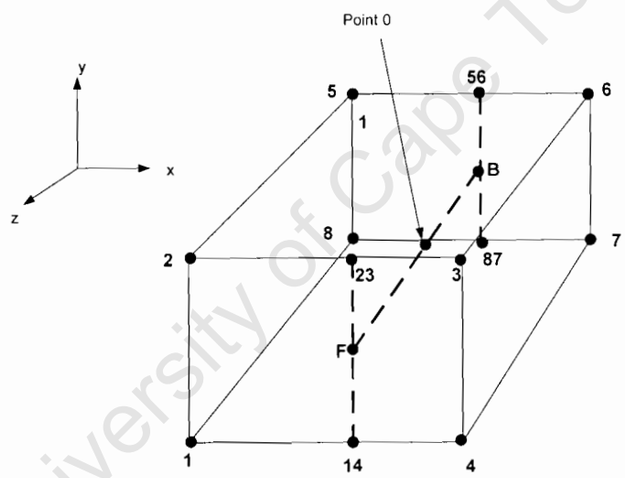


Figure C4. Tri-linear Interpolation

Note that the distance values are denoted by f_i where i is the grid point number.

Interpolation along the line *2-3:*

$$f_{23} = \frac{x_3 - x_0}{x_3 - x_2}.f_3 + \frac{x_0 - x_2}{x_3 - x_2} f_2 \text{ And if } t = \frac{x_3 - x_0}{x_3 - x_2} \text{ then } f_{23} = t.f_3 + (1-t) f_2$$

Along the line 1-4:

Since $x_1 = x_2$ and $x_4 = x_3$, then

$$f_{14} = t.f_4 + (1-t) f_1$$

Interpolation along the line *23-14:*

$$f_{23-14} = \frac{y_{23} - y_0}{y_{23} - x_{14}}.f_{14} + \frac{y_0 - y_{14}}{y_{23} - y_{14}} f_{23} \text{ If } u = \frac{y_{23} - y_0}{y_{23} - x_{14}} \text{ then } f_{23-14} = u.f_{14} + (1-u) f_{23}$$

Along line *5-6*:

$$f_{56} = t.f_6 + (1-t) f_5$$

Along line *8-7:*

$$f_{87} = t.f_7 + (1-t) f_8$$

Along line

$$f_{56-87} = u.f_{87} + (1-u) f_{56}$$

Along line Z:

$$f_{final\text{-}value} = \frac{z_F - z_0}{z_F - z_B}.f_{56-87} + \frac{z_0 - z_B}{z_F - z_B} f_{23-14}$$

Appendix D:
Distance Transforms

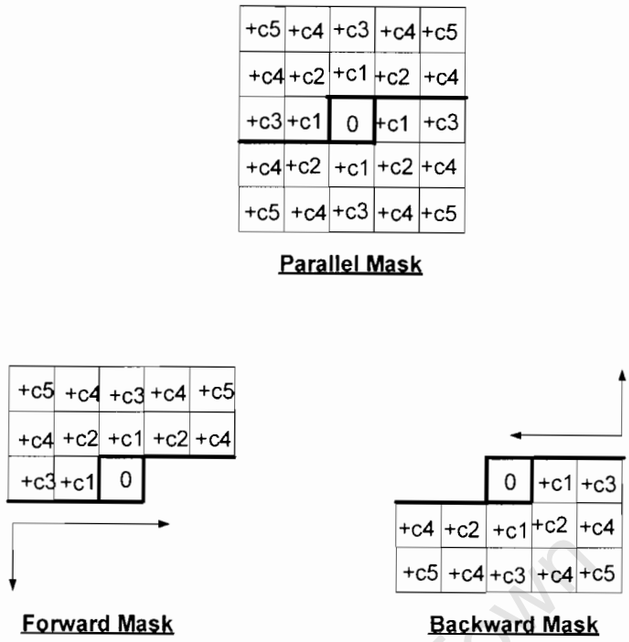


Figure D1.Distance transforms operations. Top part is the parallel "mask" and the bottom two constitute the "masks" that are necessary for sequential Distance determination.

The basic idea of DTs can be described as follows: An original binary image, to which the DT is applied consists of feature pixels/voxels with the initial value zero, and the non-feature pixels/voxels with the initial value infinity (approximately a suitably large number). All DTs are described in graphical form as "masks", see Figure D1, above. The masks constants, c_n, are the local distances that are propagated over the image. The size of the neighbourhood can vary.

The computation of the DT is either parallel or sequential. In the parallel case, the centre of the mask is placed over each pixel in the image. The local distance in each pixel mask c_n is added the values of the image pixel underneath it (including the central zero). The new value of the image pixel is the minimum of all the sums. The process is repeated until no pixel value changes. The number of iterations is proportional to the largest distance in the image. The parallel algorithm is thus:

$$v^m_{i,j} = \min_{(k,l)\in mask}(v^{m-1}_{i+k,j+1} + c(k,l))$$

Where $v_{i,j}^{m}$ is the value of the pixel in position (i,j) in the image at iteration m, (k,l) is the position in the mask (centre being (0,0)), and c(k,l) is the local distance from the mask. A small example of the parallel algorithm is shown in Figure D2, below.

The final DT result is the same whether parallel or sequential computation method is used. However, some hardware cannot handle both.

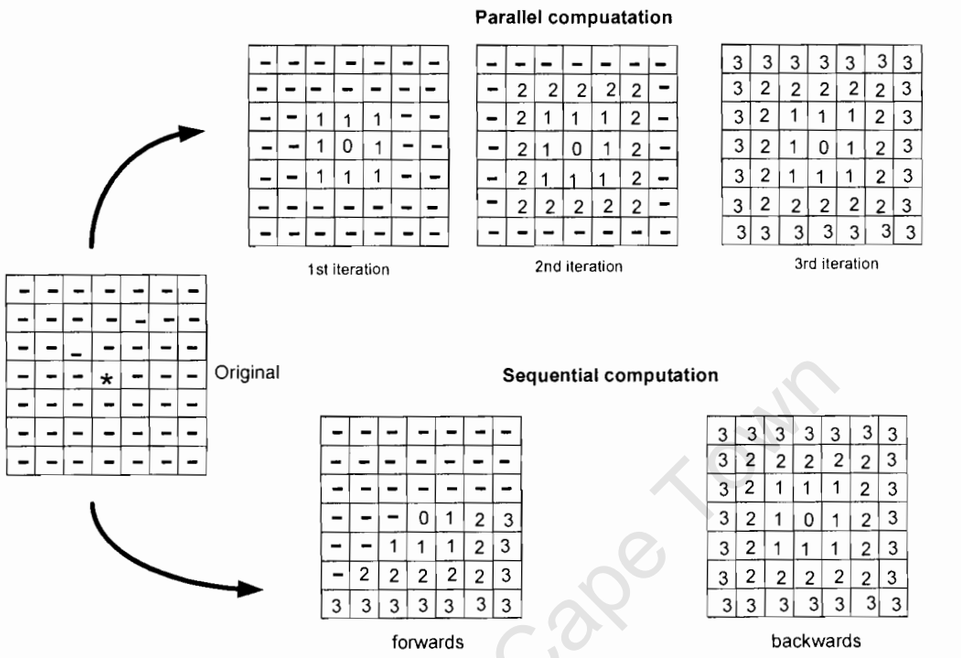


Figure D2. Computation of a DT. At the left is the original image with one feature point in the middle. The upper images illustrate the parallel algorithm. The lower images illustrate the sequential algorithm, showing the result of the forwards and backward passes.

Distance transforms can be split into the following categories (Borgefors, 1986):

- N-neighbour
- Chamfer distance transforms (CDT) and
- Vector Distance Transforms

N-Neighbour

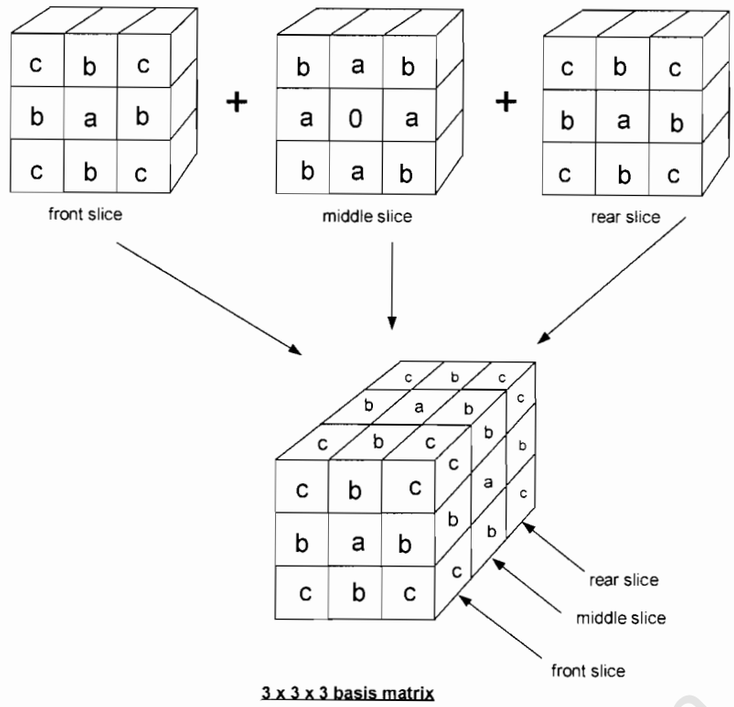


Figure D3. The basis matix from which DTs can be derived

City Block/ Manhattan: In the 2-D case, its is known as the 4-neighbour and in 3-D, it's referred to as the 6 neighbour. This is the simplest and the fastest of all DTs. It is, however also the worst approximation of the Euclidean distance (Barrow et al. 1977).The city block is described by the general 3×3 neighbourhood mask, Figure D3 with $a = 1$, $b = inifinity$ and $c=infinity$, i.e. the diagonal neighbours are ignored. The algorithms can either be parallel or sequential.

Chessboard: The accuracy for the City-block can be improved by including diagonal elements in the Distance Transformation. This defines the Chessboard DT with a=1, b=1 and C=infinity. Figure D4 below, shows the 3-D mask for the Chessboard DT.

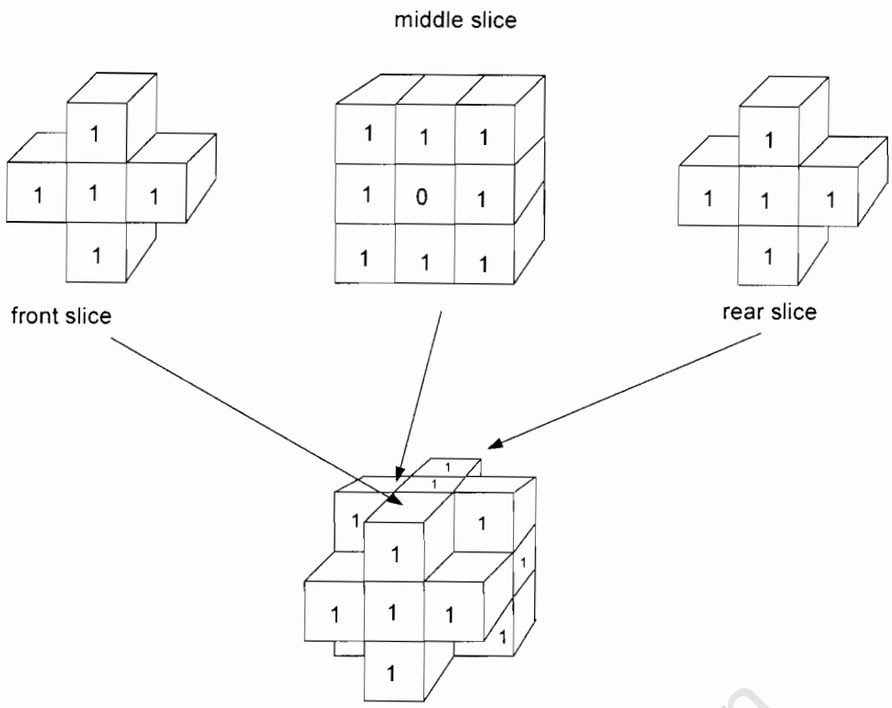


Figure D4. The chessboard DT "mask'

Chamfer distance transforms: In some cases, better accuracy is achieved by using values other than 1 for *a*, *b* and *c, to* represent the local distances (Rosenfeld *et al.* ,1966). Cuisenaire *et al.* (1999) and Borgefors (1986) have carried out extensive research work on chamfer distances. Although the accuracy is better than for the City block and Chessboard, the involvement of more elements increases computation time. The number of passes for the DT increase to between 4 and 6. Typical values that have been used include *a=3, b=5 and c=7.*

Vector Distance Transforms (VDT): Vector Distance Transforms (also known as Euclidean distance transforms –EDT) differ from chamfer distance transforms in that vector components are propagated (Figure D5, Below). Distances are calculated by evaluating the vector components once they have been propagated.

Vector distance transforms generally require more passes of the distance matrix. During each pass, the vector components are added to the necessary vector position, a decision is made as to whether any new vectors are minimal and if so, the minimal vector is stored. Vector Distance transforms were first introduced in 2-D by Danielson (1980) and implemented in a

manner similar to chamfer distance fields, computing a distance field in four passes. The superiority of the VDTs over CDTs is in the accuracy achieved (see Cuisenaire *et al.*, 1999).

(1,1)	(0,1)	(1,1)
(1,0)	(0,0)	(1,0)
(1,1)	(0,1)	(1,1)

Figure D5. Showing the VDT "mask'

Which distance metric is most suitable for Virtual urban modelling?

In some digital image processing applications, where a distance transformation is needed, it is necessary to use the correct Euclidean distance. In other cases, the distance should be as close to the Euclidean Distance as possible, but need not be quite as. If the accuracy needed is somewhat less, then the DT, which is less computationally complex, may be the best choice. Computing exact distances from the inexact features (as found in much scanned data) is not reasonable, at least not when the exact distances are much more computationally costly than adequate approximations.

The chessboard and City-block DTs can be advantageous for images consisting mainly of rectangles with the sides parallel to the co-ordinate axis, e.g., house or street scenes. Finally the city block distance is the fastest to compute, and where speed is essential rather than accuracy it may be the best choice.

In urban environments, where one is dealing with data, which is less accurate, compared to that in medical imaging, massive data sets, buildings of a rectangular and regular nature, then one would be expected to use the City block DT.