Technical Report

UCAM-CL-TR-420
ISSN 1476-2986

Number 420

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Managing complex models for computer graphics

Jonathan Mark Sewell

April 1997

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted March 1996 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Queens' College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Abstract

Three-dimensional computer graphics is becoming more common as increasing computational power becomes readily available. Although the images that can be produced are becoming more complex, users' expectations continue to grow. This dissertation examines the changes in computer graphics software that will be needed to support continuing growth in complexity, and proposes techniques for tackling the problems that emerge.

Increasingly complex models will involve longer rendering times, higher memory requirements, longer data transfer periods and larger storage capacities. Furthermore, even greater demands will be placed on the constructors of such models. This dissertation aims to describe how to construct scalable systems which can be used to visualise models of any size without requiring dedicated hardware. This is achieved by controlling the *quality* of the results, and hence the costs incurred. In addition, the use of quality controls can become a tool to help users handle the large volume of information arising from complex models.

The underlying approach is to separate the model from the graphics application which uses it, so that the model exists independently. By doing this, an application is free to access only the data which is required at any given time. For the application to function in this manner, the data must be in an appropriate form. To achieve this, *approximation hierarchies* are defined as a suitable new model structure. These utilise multiple representations of both objects and groups of objects at all levels in the model.

In order to support such a structure, a novel method is proposed for rapidly constructing simplified representations for groups of complex objects. By calculating a few geometrical attributes, it is possible to generate replacement objects that preserve important aspects of the originals. Such objects, once placed into an approximation hierarchy, allow rapid loading and rendering of large portions of a model. Extensions to rendering algorithms are described that take advantage of this structure.

The use of multiple representations encompasses not only different quality levels, but also different storage formats and types of objects. It provides a framework within which such aspects are hidden from the user, facilitating the sharing and re-use of objects. A *model manager* is proposed as a means of encapsulating these mechanisms. This software gives, as far as possible, the illusion of direct access to the whole complex model, while at the same time making the best use of the limited resources available.

Preface

Apart from some minor corrections, this report is the dissertation which I submitted for my PhD degree. For technical reasons the colour images have been replaced by greyscale images. Please note that in Figures 5.5, 5.19, 5.20, 5.26, 5.29 and 5.30 the original images were colour coded.

The research described in this dissertation has been carried out at the Computer Laboratory by the kind permission of Professor Robin Milner, the new Head of Department, and his predecessor, Professor Roger Needham.

This dissertation was typeset using LaTeX2ε and the main text consists of about 50 000 words. Most of the 112 diagrams were drawn in Adobe Illustrator 5.0 while the 58 sets of images were processed in Adobe Photoshop 3.0.

Acknowledgements

I would like to take this opportunity to express my gratitude to my supervisor, Neil Wiseman, who welcomed me into the Rainbow Graphics Group when I started and who guided me into the research which has eventually been described in this dissertation. His advice and ideas were invaluable and always freely given, and since his death last year, he has been sadly missed. My thanks go to Peter Robinson who at short notice has supervised me through the final stages of this work — his advice and comments have been very much appreciated.

I would like to thank all the members of the Rainbow Graphics Group who have ensured that my time in the Computer Laboratory has been extremely enjoyable. In particular, my work has benefitted from discussions with Oliver Castle, Neil Dodgson, Chris Faigle, Peter Brown, Uwe Nimscheck, Martin Turner and Adrian Wrigley, and I have greatly enjoyed the work I have done on the Cambridge Autostereo Display Project with Stewart Lang. The other members of the group who I have known over the past three years are too numerous to list — it is difficult to be short of ideas, enthusiasm, or constructive criticism in their company. Special thanks must go to Oliver, Neil, Peter and Chris for reading and commenting on earlier drafts of this dissertation. I must also mention the many other people who I have shared offices with and the rest of the Austin 4 wing, including Margaret Levitt and Eileen Murray who manage miraculously to keep the Laboratory running.

I am grateful for the financial support of the EPSRC during my research. I would also like to thank Queens' College for their continued support over the seven years that I have been studying at Cambridge.

Finally, I would like to thank my parents for all their support, encouragement, advice and help — I am sure they did not realise what they were letting themselves in for when they encouraged me to apply to Cambridge — and Sarah Marshall who in addition has spent four hours a day on a train for over three years so that I could stay here and do this research.

Contents

Chapter 1

Background and Motivation

— In which some problems are identified, and aims are set out.

1.1 Introduction

The last twenty years have seen a largely unanticipated explosion in the capability of computers to produce high quality, realistic digital images. Computer generated animation sequences have become common on television and in the cinema where they can blend almost seamlessly with live action. Even more progress is anticipated, since it is predicted that the power and performance available on expensive graphics workstations will appear at a low cost, on the desktop, within the next ten years.

The realism that new systems can potentially offer has encouraged interest in areas such as virtual reality, architectural walk-throughs, computer-aided product design and scientific visualisation, and has also made a huge impact in the entertainment industry. Most of this interest is based around creating images of three-dimensional "worlds", whether they are modelled on the real world or are an abstract representation of the results of a numeric calculation.

This dissertation studies how systems designed to process these three-dimensional (3D) models will have to evolve in order to fulfil users' demands in the future. In particular, it argues that there will be a continued increase in the size and complexity of the models that users wish to view and interact with, and that new techniques will be needed to facilitate both the creation of models and their use.

1.1.1 Synthetic Image Generation

The process of synthetic image generation is usually described as consisting of two stages (Figure 1.1) — first a **model** of the scene is constructed within the computer, and then an image, or set of images, is created by **rendering** the model from a particular viewpoint.

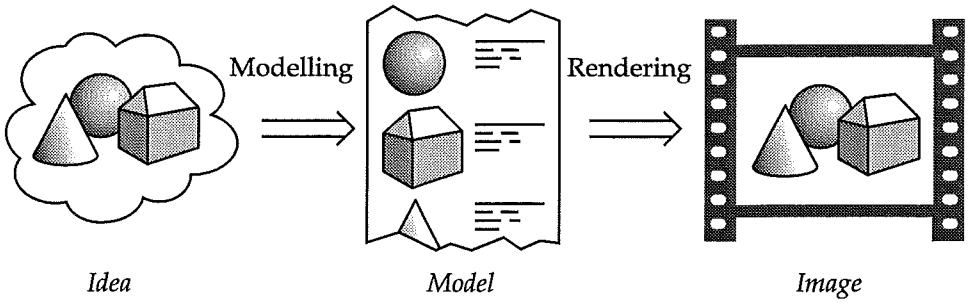


Figure 1.1: Modelling and Rendering.

Although much of the research hitherto undertaken in computer graphics has been concerned with improved rendering techniques in order to generate more realistic images faster, the emphasis is shifting towards improving the modelling process itself. Models may be generated manually by a designer, automatically by the computer in response to a user's commands or more commonly now by automated input devices such as 3D scanners, or from the results of numeric computations.

While computer imagery is only slowly becoming accepted as an art form, computers have made a significant impact on the design world, mainly due to their ability to sustain repeated and accurate editing and alteration, thereby increasing productivity and quality.

One of the main attractions of 3D computer graphics is the ability of the user to control where the world is viewed from. This interactivity has only recently become possible on moderately priced equipment. The desire for interaction is also extending to the modelling process, which is no longer the sole domain of the expert designer. Instead, users wish to create their own 3D worlds and explore them — these two stages are no longer separated. Computer graphics systems can, unfortunately, still be difficult to control, slow to use and inflexible. These problems need to be addressed in order for computer graphics to become more useful.

1.2 Three-Dimensional Models

Modelling, as noted in [**Heckbert87**] [**Quarendon93**] and [**Hall91**], is often the bottleneck in the production of synthetic images. This section examines how computer graphics models are created, and the difficulties associated with creating complex models.

1.2.1 Models and Databases

Computer graphics models often start from a real world entity which the designer wishes to re-create in the computer. An internal conceptualisation of this model will need to be translated into a description based upon geometric quantities, such as polygons, curved surfaces or density data. This information will have to be stored, both internally to an application that uses it, and more permanently on a file system. These stages are summarised in Figure 1.2.

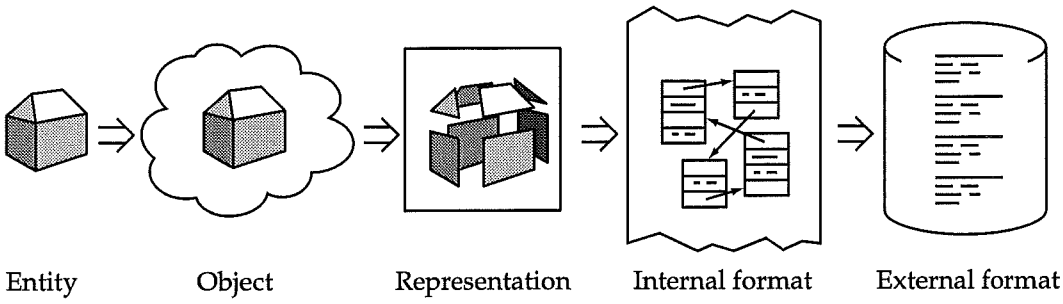


Figure 1.2: Some of the states which a model passes through during creation.

The term **object** will be used to describe the notion that the model creator has of the entity to be modelled. This is in essence an abstract concept, and as such cannot be directly and meaningfully represented in the computer. Such objects might be described in human terms as "an armchair" or "the Eiffel tower", but this does not give the computer sufficient information to draw a picture of them, unless of course it has already been told what an armchair should look like.

In order to do something with an object other than just refer to it, the computer needs a **representation** of the object. This would describe the original object in terms of understood primitives, such as polygons or spheres, or perhaps in terms of the sub-objects from which it is constructed. There is no reason why more than one representation of an object should not exist. For instance, one rendering system might use polygons as its only input type, while another would use curved surfaces, and a third would need a volume-based representation. All of these, however, describe or represent the same underlying object.

Inside the modelling or rendering software, a representation of an object will be stored in some structure(s) in memory – an **internal format**. When saved on a storage device, these

will typically be written out in some **external format**. Many formats can be used for describing the same data. These may be human-readable, such as the Rayshade scene description language [**Kolb91**], or only machine-readable, such as the Open Inventor binary data format [**Wernecke94**]. Typically, the closer the external format is to the internal one, the quicker it is to parse into memory, while the larger the external format, the longer the data takes to move around. The external format is usually determined by an **object description language** (ODL).

For a given representation of an object, there are many different formats in which it can be described. By the time the object has reached an external format, there may have been some loss of information. For instance, a polygon mesh model might be written to a file as a list of triangles, discarding the connectivity information. Although the geometric description might be considered to be equivalent, this could make processing less efficient. This is usually because the ODL is closely related to the internal format, which is derived from the needs of the rendering system rather than from a need to describe the objects being modelled in a way suitable for a variety of different systems.

One final important distinction is between **implicit** and **explicit** representations. The data actually processed at the final stage of the system is the most explicit representation. This data may be described more succinctly by an implicit representation that provides sufficient information in order to be able to *generate* the more explicit one, but is probably much more concise. A **generator** is required to convert the implicit representation to the explicit one, and will involve a time penalty. An external format is much more likely to contain implicitly defined objects since this will be more compact, thus reducing storage size and transfer times. Additional processing will be necessary at load time or rendering time depending on when most expansion is performed.

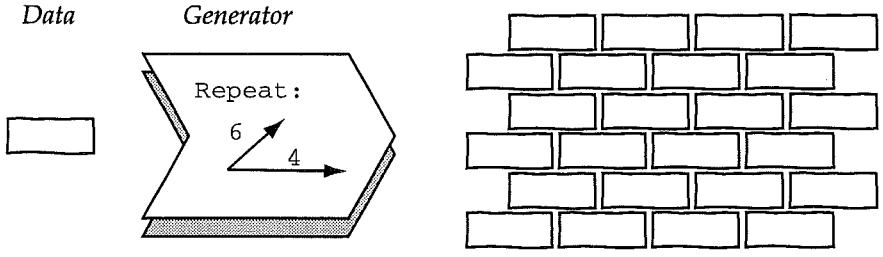


Figure 1.3: Generators create an explicit representation from an implicit one.

As an example, consider a simple brick wall (Figure 1.3). The explicit representation would contain a block at the location of each brick. An implicit representation might describe the wall in terms of an array of six rows with four blocks in each row. The latter is obviously more space-efficient, but current systems are more likely to cope with the former.

1.2.2 Model Creation

The widespread use of graphical design tools in the production of 3D models for computer graphics has started only recently. Previously, object specification was usually textual, and indeed in many cases, this is still the approach anticipated by the designers of rendering systems when specifying object description languages.

The textual approach was to a large extent due to the impracticality of graphical feedback — rendering a model during construction was too slow to be useful. A preview facility was often provided to generate rough images of objects, but still was not fast enough for interactive editing. Modern workstations are capable of providing at least a wireframe real-time editor for modestly complicated scenes, allowing direct interaction and manipulation. Today, this would be the preferred mode of design.

Entering the data for models is still a difficult and time-consuming task. Recently, hardware devices such as 3D scanners and digitisers have come into use for the rapid creation of models. These have their own associated problems, the principal one being the huge amount of data that they produce [**Turk92**].

Manual data entry has been aided by the development of constraint-based editors and higher-level modelling paradigms. Generative modelling [**Snyder92**], for instance, attempts to describe objects in terms of the operations needed to create them, rather than the primitives that make up the final representation. This makes editing and higher level operations much more plausible. Similarly, a number of algorithms have been described which create models, particularly of natural objects such as plants [**Prusinkiewicz88**], by a growth process, again allowing the object's specification to be abstracted from the actual geometry used to represent it. Enhanced procedural models [**Amburn86**] can be used to construct models satisfying constraints, and a rule-based approach has been applied to domains such as house construction [**Heisserman94**]. Physically-based modelling systems are also being developed which allow the construction of models that obey physical laws (for example [**Snyder95**]), but these rely on solving difficult problems such as collision detection.

The trend towards the specification of objects in logical terms rather than by giving the details of a particular physical representation offers a number of advantages. Modelling and editing should become simpler, since large scale or higher level features can be manipulated individually to avoid having to modify many low level features consistently. Variations of the object can be created easily; a chair model, for example, could have several instances with different height backs, different thickness cushions, or different colour frames (Figure 1.4). Finally, it is simpler to have one higher-level representation from which several specific representations are created for use in systems with different requirements .

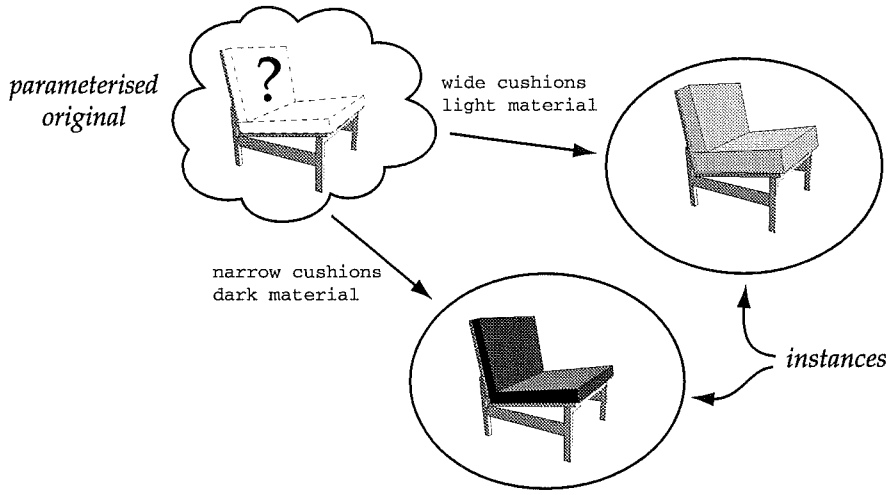


Figure 1.4: A simple parametrised description of a chair together with generated instances.

1.2.3 Object Sharing

If the computer modelling process is compared to the corresponding real-world construction process, several differences become apparent. One of the most noticeable is the lack of pre-constructed parts in use. While manufacturers use raw materials to some extent, they will make use of pre-processed materials and components, and also of complete objects chosen from a wide range of alternative designs. If new computer objects could be readily constructed from existing ones, the length of the design process would be reduced and the quality of the models could improve. This idea is being exploited in the programming world, where the complexity of modern software products is enabled by the use of interface-specified libraries, and object-oriented programming techniques.

Sharing and re-using computer graphics *models* is much more difficult. Repositories of objects, and companies that distribute models, are starting to appear, but this approach is still in its infancy. One problem is the lack of choice in the range of items available. This can only be solved by allowing a wider base to build up over time, and by encouraging the use of higher level modelling techniques which possess the ability to describe a variety of object instances.

A more significant problem is that of incompatibility. Objects are typically described by *one* representation in *one* particular ODL. Many sources offer objects in a few different formats. As noted above, whilst essentially describing the same object, these may have a number of differences due to restrictions in the ODL or due to the means of conversion between them. Transformations between different formats are often achieved by using file translators, but many translators will either have to miss out some data or generate values for information required by the target format not present in the source data. This is time consuming and

may lead to inefficiencies and problems later (*e.g.* [**Erikson95**]).

Differences in representation are much harder to overcome. Converting models from a constructive solid geometry (CSG) representation to a polygonal one or from polygonal to volume data can be done, but rarely is. In order to produce a different representation an object modelled in one representation is quite likely to be re-modelled rather than converted. While a conversion might produce lower quality results, any result would be a good starting point.

One solution to some of these portability questions is standardisation. This is conceptually appealing, but it is not a practical proposal for a number of reasons. Primarily, it would be difficult to convince a large proportion of the community to adopt such a standard at the expense of those already in use. Many of the languages for describing models have been released with the intention of becoming the standard, but few have been widely adopted. Standards that *have* succeeded, however, have been based on widespread use of the packages that support them. New types of objects, and new requirements for representations, are continually appearing. It is impossible to anticipate these in advance, so an *extensible* system is necessary. This leads to difficulties deciding who specifies the extensions to the language in order to prevent different dialects appearing, and a large cost is incurred when software has to be updated to cope with new constructs.

1.3 Rendering

Currently, the main purpose of processing models in 3D computer graphics is to create images of them, communicating information from the modeller to the viewer. Until recently, image synthesis has concentrated almost entirely on "photo-realism" — the quest to reproduce, on the computer screen, the world as it would be seen by a camera. Recently, there has been renewed interest in using computer graphics for artistic ends, and this section examines the requirements of different image generation paradigms.

1.3.1 Realism

The drive for realism has been motivated by both commercial and artistic reasoning. If designers can use a computer to generate an image of an object without having to physically construct it, the design process can return greater value for money. Modelling the real world can also be viewed as the greatest possible challenge, since, being the most familiar, it is the hardest to fake.

The illusion of virtual reality (VR) is based on the suspension of disbelief in the user. To achieve this, a VR system must provide images that users could expect from their experience of the real world. The popularity of scientific visualisation techniques is due to the basis of their images in the real world. Humans have extensive experience of understanding

solid three-dimensional shapes, so by mapping abstract data to something more concrete, the unfamiliar can be visualised in terms of the familiar.

When looking at a real world scene, the image received is usually characterised by its *complexity*. Digital images, on the other hand, generated from synthetic models, have been easy to identify due to their relative simplicity, a feature of both the modelling and rendering stages. Processing power and modelling techniques now enable realistic images to be generated, but still at great cost and usually only for a limited set of subjects. Computer graphics algorithms have also progressed from simple modelling and rendering using flat shaded polygons to complex physically-based systems such as ray-tracing and radiosity, which are slowly reducing the number of optical effects that cannot be simulated in the computer. Unfortunately, costs increase significantly with the realism of the rendering process.

1.3.2 Non-Realistic Rendering

As more and more realistic images are generated, some non-realistic rendering techniques have become more prominent. Initially concerned with producing an "artistic" image, such as the painting effects of **[Haerbili90]**, or **[Lansdown95]**, some techniques are now used to aid comprehension.

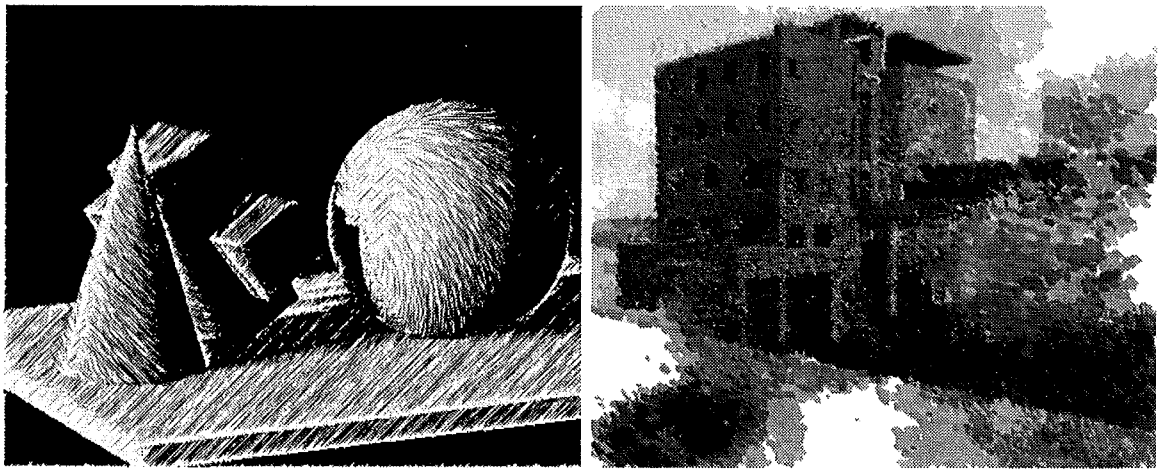


Figure 1.5: Some examples of computer-generated non-realistic renderings *(images from the Piranesi System developed at the Martin Centre CAD-LAB, Cambridge* **[Lansdown95]** *).*

One of the challenges of artistic rendering algorithms is to obtain the higher level information from the model that a person can instantly gain from observing a real scene. Such information is often central to comprehension. One example is the recent interest in imitating pen-and-ink illustration techniques **[Salisbury94]**. It was noted that these hand-drawn illustrations were still used in many technical manuals in preference to photographs, since

there was a natural filtering of information, and emphasis was given to certain elements in preference to others. The clean drawing style smoothes over the small detail so that it does not swamp the viewer with unnecessary information, while texturing techniques are used to indicate the *presence* of detail which in itself can be a vital clue [**Saito90**].

All hand-generated computer graphics models include an artistic element, since object-modelling is essentially a human-based simplification task. Out of necessity, early models were merely simple representations of objects. With only wireframe display techniques and no hidden surface removal, complexity produced unintelligible results. In these cases, the model creator actually performs some of the comprehension tasks for the viewer by modelling only those details considered important.

1.3.3 Interpretative Rendering

Some of the early graphics techniques, such as depth cueing when wireframe rendering, employed because of the relatively low power of the machines in use, can actually increase the information conveyed to the viewer; the user can be forced to concentrate on different aspects of the image at different levels. Thus attention can be drawn towards some objects, and away from others [**Strothotte94**] [**Winkenbach94**].

For instance, an electrical engineer needs different information from an image of a building to a plumber, and both of these will be very different to an interior designer's requirements. This cannot be achieved through modelling alone, since all the details must be present in the model. Rather the renderer, under the control of the user, needs to pick out which features to emphasise and which to hide in a given context [**Hubbold93**] [**Selgmann91**].

Other examples include 3D filters or **lenses** such as the "MagicSphere" [**Cignoni94**]. These are objects within the model under the viewer's control which affect the manner in which objects near to them are drawn. The user can move the lenses around to examine different areas without overloading the information present in the image. Lenses that display more detail, or display only certain types of objects are to ways in which to provide user-friendly tools.

Such ideas are essentially a combination of realistic rendering with the interpretive aspects of artistic rendering. The tools described in this dissertation will be useful in enabling a new paradigm, which might be called **interpretative rendering**, providing the user with selective and context-dependent control over the levels at which information is presented in an image, and for which computer graphics is uniquely suited. Interpretative rendering is characterised by the use of advanced techniques to *help* the viewer interpret the model rather than merely providing a realistic image. This might appear to exhibit a desire for simplicity, in contrast to the drive for complexity associated with realistic rendering. The key however is the *selective* use of complexity. While complexity must be present in the model, it is important to have tools that allow it to be used only when necessary. It is this concept that underlies this dissertation.

1.4 Complexity

The most obvious trend in the development of computer graphics has been the increase in complexity — both the complexity of the models (the data) and that of the rendering systems (the algorithms) that use them. As long as new machines are developed that are faster than existing ones, and have the capability of increasing the complexity of the results calculated, this trend is set to continue. If this is the case, then it is important to examine how systems will scale as this complexity increases.

Rendering algorithms exhibit an obvious increase in complexity from simple point and line drawing, through hidden surface polygon rendering, to ray tracing and radiosity techniques (*e.g.* [**Foley90**]). With this increase in complexity comes an expansion in the associated computational and memory costs. Even more complex techniques have been proposed, which can render more physically realistic effects such as caustics or polarisation, but are kept back from general use because their costs remain prohibitive.

The trend in model complexity is also simple to trace. Early synthetic images were often of a few objects constructed from basic mathematical shapes. Driven by the quest for realism, today's scenes contain large numbers of realistically modelled objects. This expansion has been possible due to the ability of rendering systems to cope with more complex models, and the ability of designers to create them. Such an increase in the desire for model complexity accentuates the bottleneck in the model construction stage. Since it remains the most significant point of human input, it is a process that needs to be the focus of attention.

1.4.1 Complexity and Computation

The assumption underlying this discussion is that complexity entails computation, that computation entails time, and that there will always be a desire to do things faster. Speed is considered important for a variety of reasons. If a system is to be interactive, then it must be able to generate images certainly at no less than about ten frames per second, otherwise the delay begins to impair the user's perception [**Funkhouser92**]. If machines become available that can run faster than that, this does not mean that other issues are still not worth pursuing. The additional speed might enable more complex data to be processed to achieve higher quality results, or allow extra computational effort to be spent elsewhere.

The costs of complexity do not just become apparent in the time it takes to render an image. A complex model will probably take up a significant amount of storage space, both internally in computer memory, and on an external storage device. A higher level modelling language might allow for compact storage of models, but this cannot be relied upon, and there is a penalty in terms of the complexity of the system required to use such languages. Together with storage space, go the time and bandwidth required to communicate the model from the storage device into the computer memory, as well as to parse it from the external stored format into the internal format required by the application (Figure 1.6).

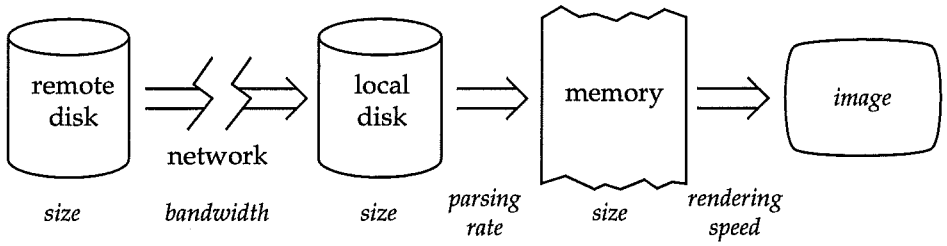


Figure 1.6: Different parameters affecting data size and delay when obtaining and displaying a model.

It could be argued that many time and space limitations can be solved by waiting for a larger, faster computer system. Many users will always lag behind in terms of equipment, but this should not preclude them from using the models that have been developed. They might not be able to expect the same interaction rates or the same quality of image at the end, but results should still be obtainable. Furthermore, they should be able to control the quality at which they require the result in order to make the best use of their current equipment.

1.4.2 Current Complexity

Rendering systems are now reaching a stage where real-time, realistic image generation is possible. For instance, the Silicon Graphics Reality Engine^2 (RE^2) graphics system is one of the most powerful commercially-available Z-buffer based polygon rendering systems. It claims to be capable of almost a million anti-aliased texture-mapped triangles per second [**SGI93**] for a triangle mesh whose elements have an average area of 10 by 10 pixels. At 25 frames per second, this reduces to 40 thousand triangles per frame — enough to cover a high resolution screen twice. Lower end workstations manage between one and two orders of magnitude less than this, and non-specialised machines much less again.

The most compact format for these triangles would be as a single mesh needing roughly one three-dimensional position per vertex. This would yield almost 500 Kbytes of data, which would be doubled if vertex normals were also stored. More realistic model storage would involve several times this memory space for similar complexity. The machines using the RE^2 graphics system operate with around 256 Mbytes of main memory, shared between several processors in order to provide the capability of feeding the graphics system at a sufficient rate. Storing local copies of databases which run into many megabytes places a high load on disc systems and requires a high bandwidth path from disc to memory.

The geometric compression scheme proposed in [**Deering95**] reduces the accuracy of some geometrical information, and may be useful for reducing sizes of data for transfer over slow networks. Realistic transfer rates over the internet are rarely over 1 Kbyte per second yielding download times in the order of minutes for moderately complex data sets. Although

local networks can provide higher speeds, this bandwidth is not increasing as fast as storage capacity [**Read93**]. Compact higher-level representations which are expanded locally into lower level data would also reduce network load.

Datasets covering more than can be seen in a single image are correspondingly larger. Digital elevation data can be stored very efficiently, but even then the height data for England would occupy 500 Kbytes if sampled on a one-kilometre grid. If finer resolutions, texture maps, and other objects are to be stored as well, then this could rapidly expand by an order of magnitude. Increasing the resolution to 10-metre intervals would produce 50 Gbytes of data. The Terravision project [**Leclerc94**], for instance, uses a dedicated gigabit ATM network in order to obtain terrain data from different locations around the globe.

1.4.3 Model Detail

The trend for greater complexity looks set to continue. The use of computer graphics as a tool is growing, and is being applied in more diverse situations. The popular press view of computer graphics describes models of more complexity than can really be contemplated.

Current models tend to fall into one of two categories. The first of these exhibits the "film set" phenomenon [**Heckbert94**], in which the model has been designed so that it looks detailed from a given point of view, but given freedom to look around the viewer will find that all other aspects have only been sketched in at best (Figure 1.7). This fails to cater for the freedom and interactivity that constitute the strength of computer simulation. The second category trades visual complexity for interactivity. The user can navigate through a larger area which is only represented roughly — whatever the viewpoint, the results are equally disappointing.

The range of scale visible in an average real world scene is quite large. When close, the eye is quite capable of seeing detail of the order of a tenth of a millimetre; in the average scene, objects will easily appear up to several metres away. Freed from the conventions of physical reality, the viewer is capable of pushing these boundaries further. The T_Vision system [**TVision95**], for example, demonstrates a visualisation application that allows the user to zoom smoothly from viewing the entire planet to looking through the window of an office. However, there is no reason why the viewer might not wish to continue further. The result is a huge range of scales at which the viewer might wish to examine the model.

The human eye has its own in-built means of reducing complexity – detail is only visible in a very small central portion of the image on the retina. The brain manages to build up a picture of a complex world through the interactive viewing of many distinct small regions of detail. This offers a glimpse of how a computer system might similarly manage complexity. By mimicking this selectivity, it may be possible to reduce the overall complexity in the system to a manageable level.

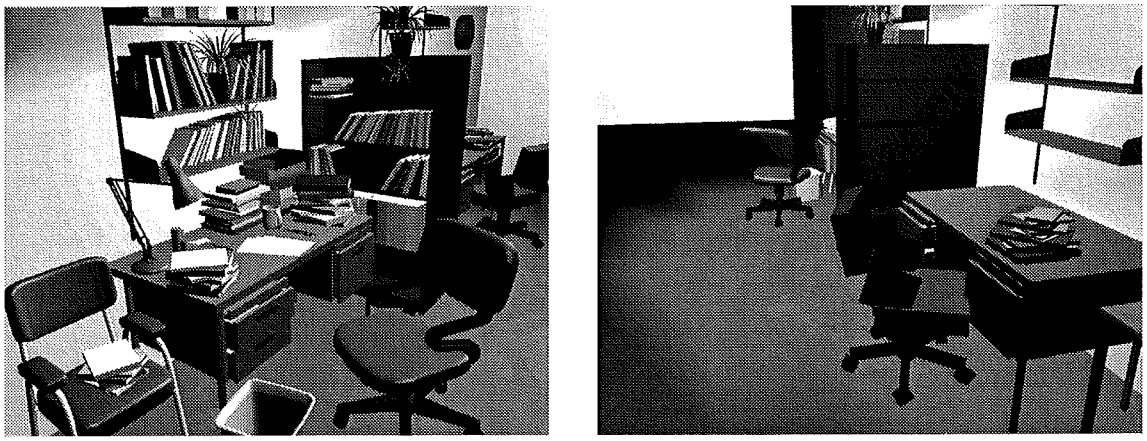


Figure 1.7: Purpose-designed models are balanced around a few viewpoints *(left)*, but have insufficient detail when they are seen from other viewpoints *(right)*.

1.5 Problems and Aims

Having examined the development of computer graphics and identified likely paths that it may follow in the future, this section looks at the problems facing future computer graphics systems. A set of aims is presented that should be addressed when constructing systems in order for them to function in the future. These aims are illustrated by an example application which can be used as a concrete example in the remaining chapters.

1.5.1 The Problems

In 1990, in an essay on the future demands of computer graphics, Donald Greenberg wrote

> " I anticipate a vast increase in model complexity ... we will move from the design of elements to components to assemblies to systems ... The size of typical modelled environments ... will increase by two to three orders of magnitude in geometric complexity alone. " **[Greenberg91]**

To support this efficiently, it will not be sufficient merely to require faster computers with larger storage capacity and better input capabilities. This would not only be ignoring the difficulties associated with the construction of databases and the ability of humans to process the volume of information being thrown at them, but also the power wasted in extending systems that are balanced around current levels of complexity.

Systems need to be designed around the needs of the users. For computer graphics systems, this is not only the viewers but also the model designers. The introduction of computers is often the enabling technology that allows non-specialists to start experimenting creatively in new areas and computer graphics users will probably wish to be able to design their own worlds. The evidence of this can be seen in the desktop publishing world where the capability of the general public to use tools has released publishing from the domain of the specialist into the hands of the amateur. It will also increasingly be the case that these two groups of people will converge. Designers are already users, but more users will also wish to be designers, if perhaps at a slightly different level.

For the users of graphics systems who wish to see the images created by others, there is one over-riding consideration – ease of use. The technical expertise required of the user should be minimised in what is essentially a non-technical process. At the same time it is important that the ability to control the graphics system is there for those needing this facility.

Similarly, for the model designers the emphasis must be on making the task easier. If complex models are to be constructed to match demand, as many hindrances should be removed as possible. More precisely, it is important that any difficulties are directly related to the task which is being performed, namely model design, rather than arising from secondary areas, such as data formats, or considerations about the systems with which these models will be used.

1.5.2 Aims

In order to solve these problems, seven high-level aims are identified for systems that can be used to create and process large and complex 3D models. Aims 1-4 are clearly driven by the need for flexibility in modelling, while aims 5-7 focus on the needs of the end user when using the graphics system as a tool.

1. **Scalable**
   The system must have the ability to cope with any size of model.
2. **General**
   The model must be able to contain a wide range of objects and be usable by a wide range of applications.
3. **Flexible**
   It must be possible to specify the objects in a wide variety of ways.
4. **Distributable**
   It must be possible to have the model distributed over many physical locations.
5. **Controllable**
   The system must be controllable by the user in a straightforward way in terms of basic quantities such as quality or time.

6. **Consistent**
   Control should be applied equally throughout the system so that different parts of the system behave in a consistent manner.

7. **Portable**
   Solutions must be proposed that allow a system to function on a wide variety of hardware with differing performance characteristics.

These aims emphasize the management of the model, existing separately from the particular renderer or application, thereby producing **model-centred** graphics systems. This new approach is in contrast to the traditional application-centred models. Such an approach is a necessity if complex models are to be created and maintained, since creative effort, and management time and space cannot be wasted. Effort must be re-used as much as possible to minimise the costs by spreading them over many situations — this is the key to model scalability. By fulfilling these aims, the exchange and use of models can be vastly simplified. It will not *remove* the need for converting object types or modelling from scratch, but will reduce the dependence on them that currently exists.

Standardisation is *not* included amongst these aims since it is considered to be impractical. Nevertheless, standardisation has many appealing characteristics, and approaches that do not advocate a standard should bear these in mind. In particular, whenever data is processed from one form into another, there is likely to be both a loss of fidelity and an overhead incurred. A realistic alternative is to *encourage* the use of only a restricted number of representations, but also to design systems that are *capable* of dealing with the differing representations that will still occur.

Together with discussing a separate model, it is necessary to consider how this model will be used by many different applications. As with object types, it cannot be expected that a standard rendering system can be preferred in all circumstances. Instead it must be made simple for existing and future applications to accept and manage data from any model without placing an undue burden on system constructors.

The leap in complexity of the models must not produce a similar leap in the complexity of the system as far as the user is concerned either. In fact with increased data complexity, users will appreciate a simplified interface to enable them to concentrate on more important matters than, for example, the coefficient controlling the number of secondary rays to direct at light sources whose intensity is less than some user-defined constant!

1.5.3 An Example Application

To illustrate these aims, and also the following chapters, a modelling challenge will be proposed, in a similar vein to the **Geoscope** described in **[Read93]**, together with a description of two machine configurations for which this model is appropriate. The hypothetical ap-

plication will be constrained by the power of two systems on which it might run in order to provide a test for the aims presented above.

The Systems

The first rendering engine is chosen to be as familiar as possible — a simple polygon system with capabilities for texture mapping and lighting effects, constructed around a Z-buffer. This is fairly standard for systems designed for real-time rendering and would have similar capabilities, for example, to the standard OpenGL rendering library [**OpenGL93**]. This can be run on a powerful computer with graphics hardware, but should also work on a smaller computer with a much lower graphics output capacity.

The second system is a smaller, slower one, which does not produce images fast enough for real-time animation. Instead, it is used over longer periods of time to generate high quality images sequentially off-line using rendering methods such as ray-tracing.

The Model

A straightforward example is chosen that can be easily described and elaborated upon — a model of a city. This model will be distributed over a network, with different items stored at different locations in order to distribute storage costs and access times. These locations will be linked to the systems under consideration by variable capacity links. The distribution mechanism is not important, only its impact on apparent performance.

The model will contain a wide variety of objects, from terrain data and plant life to buildings with furnished rooms. As expected, these will be stored in a number of different formats, and contain objects with different types of representations, from polygon meshes to volume data to particle systems. Not all objects will be stored explicitly; some may be generated as needed. For instance, houses in an estate might be generated automatically from the same structural framework, but include different furnishings in different locations.

The aim is to describe systems with which the user is able to explore this city, to look inside any of the buildings or other areas that lie within it and examine objects at any level from an overview of the entire city, to an item in one of the rooms.

1.6 Summary

This chapter has argued that there is a need to take a new look at the way in which computer graphics models are treated. This is necessary if graphics systems are to be able to fulfil the future requirements.

The central theme of this dissertation is that of a new model-centred approach to 3D computer graphics. The next chapter describes how such an approach can fulfil the aims above, and also introduces the new ideas that are necessary to make such an approach feasible. Each of the following chapters examines a different aspect of the solutions proposed, while the final chapters draw all these strands together and analyse what has been achieved in the process. A more detailed overview of this dissertation is given at the end of Chapter 2 after the central ideas have been introduced.

Chapter 2

Model-Centred Systems

— In which mechanisms are introduced to enable a model-centred system to be constructed.

This chapter considers the problems facing a system that attempts to meet the aims drawn up towards the end of the preceding chapter. It examines exactly what data is required at any one time in a graphics system and indicates how the volume of data that *needs* to be processed can be reduced. If this can be done to arbitrary levels, then the incompatibility between the volume of data in the model and the volume that can be reasonably managed by a given application can be resolved. A framework is described which incorporates these ideas and within which the goals of Chapter 1 can be achieved. This enables the model to exist independently from the applications that use it, resulting in model-centred, rather than application-centred, graphics systems.

2.1 Visibility

Progress would be difficult if *all* the data forming the model was necessary for *each* calculation performed by the application. Computer graphics applications are fortunate in only requiring a limited amount of data when performing a given calculation. Most calculations are restricted by *visibility*, needing to act only on the parts of the model that are visible from some region of space. For large models, this visible set should only be a small proportion of the total size.

When generating an image from a model, if only those objects in the model that have an

impact on the final image were selected in advance, a **model subset** would be formed that would be sufficient for the application to draw an image at any one time. Furthermore, as the visibility requirements change over time, it is reasonable to expect that the model subset required would only change slowly. This phenomenon is a result of many types of coherence and has long been exploited in computer graphics.

2.1.1 Visibility Queries

Resolving a visibility query involves rapidly determining which items in the model are possible solutions. The most common example of these queries is in image generation, but lighting contributions, shadowing, or radiosity computations all involve solving essentially the same problem.

Ray-tracing [**Glassner93**], for example, attempts repeatedly to answer the question "what object can be seen from a given position in a given direction?". Because those objects that are not visible play no part in the immediate calculation, they can safely be removed from consideration. For a Z-buffering system [**Foley90**], the objects that are not visible in the final image can again be ignored in the computation.

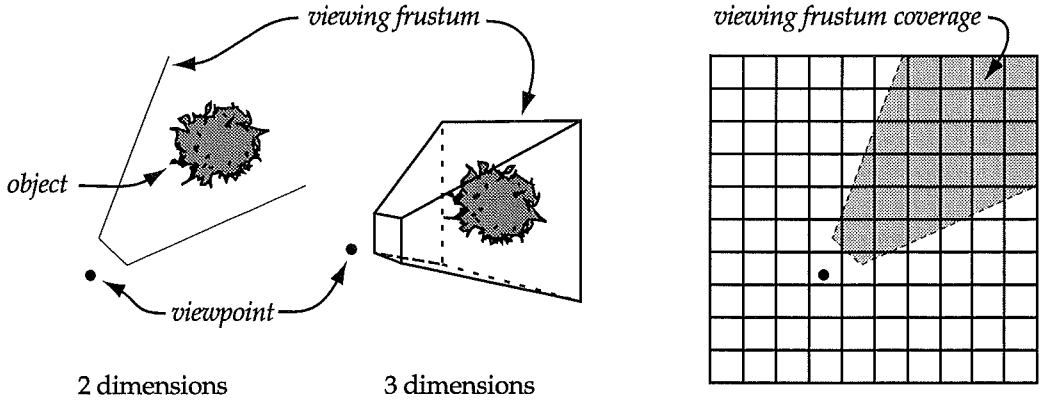


Figure 2.1: The viewing frustum in two and three dimensions, and an illustration of the percentage coverage of a two dimensional scene by a viewing volume.

When generating images of small scenes consisting of a single object which is being examined from different angles, most of it will be visible all the time. For a large scene, however, where the viewer is *immersed* in the model, only a small part will be visible at any one time. The **viewing volume** (or **frustum**) is the volume in model space which projects onto the image. The objects that fall outside this region cannot be directly visible in the final image (Figure 2.1 *(left)*). In an evenly populated three-dimensional volume, with the viewer at the centre, only a small fraction of the volume (about $1/30$th for a large 45° field of view) falls within the viewing frustum at any instant. It should be noted that many models (such

as landscapes) are more roughly two-dimensional where the fraction grows noticeably to about 1/8th (Figure 2.1 *(right)*). Many systems use a crude **culling** stage in advance to construct a subset of the model by removing objects that are guaranteed not to fall inside the viewing frustum [**Mueller95**] [**Falby93**].

2.1.2 Visibility Classification

Currently, the most common visibility determination method is the Z-buffer, due to its ability to resolve complex visibility situations in a straightforward and hardware-friendly manner. The process is described in Equation 2.1, where objects from the entire model $\mathcal{M}$ are passed through the object culling stage to remove the set $\mathcal{C}$ of culled objects, leaving a set $\mathcal{Z}$ of objects to be considered (Figure 2.2). These are then passed through the Z-buffering stage to identify which objects and parts of objects are actually visible in the final image — namely the **visible set** $\mathcal{V}$.

$$\mathcal{M} \xrightarrow[\text{cull}]{} \mathcal{M} \setminus \mathcal{C} = \mathcal{Z} \xrightarrow[\text{Z-buffer}]{} \mathcal{V} \tag{2.1}$$

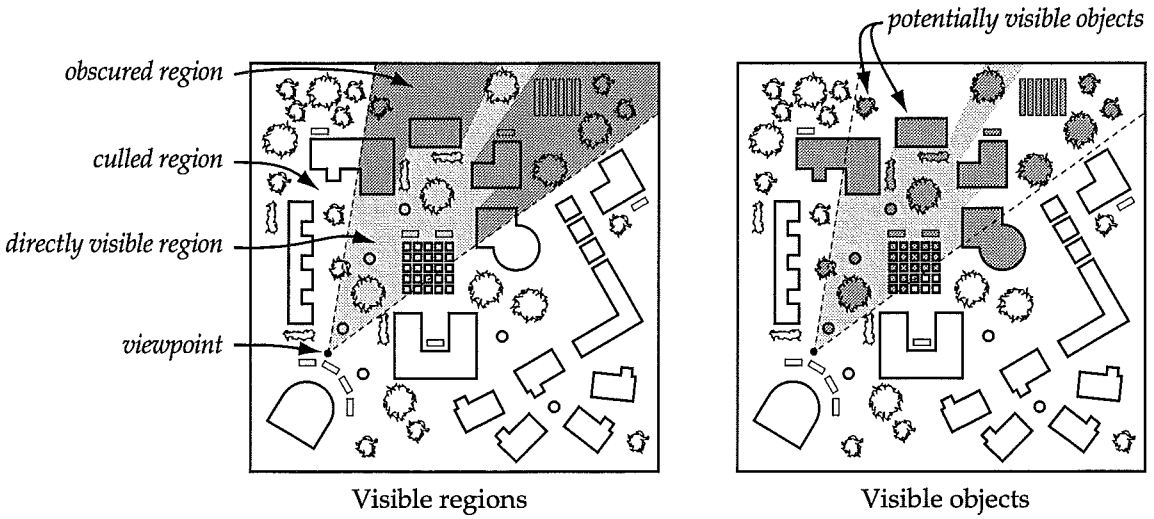


Figure 2.2: A plan of an example model with the visible and culled regions of space marked *(left)* and the set of potentially visible objects *(right)*.

Ray-tracing a *single* ray is a very similar operation, except that culling now removes the objects that cannot possibly intersect the ray, so for each ray R,

$$R : \mathcal{M} \xrightarrow[\text{cull}]{} \mathcal{M} \setminus \mathcal{C}_R \xrightarrow[\text{intersect}]{} \mathcal{V}_R \tag{2.2}$$

A *complete* ray-traced image is formed from many applications of this basic ray-tracing operation to calculate visibilities at each pixel, but also as part of the lighting model which

may involve tracing rays to light sources and reflected or refracted rays [**Glassner93**]. The total visible set for the entire image is defined as

$$\mathcal{V} = \bigcup_{\text{rays } R_i} \mathcal{V}_{R_i} \tag{2.3}$$

and the culled set for the entire image as

$$\mathcal{C} = \bigcap_{\text{rays } R_i} \mathcal{C}_{R_i} \tag{2.4}$$

$\mathcal{V}$ will contain *parts* of many objects where the whole object is not visible. A more useful set is the **visible object** set $\hat{\mathcal{V}}$ consisting of the objects which appear in part in $\mathcal{V}$. It is thus the smallest set of objects which is sufficient for rendering the image, so that

$$\mathcal{M} \supseteq \mathcal{Z} \supseteq \hat{\mathcal{V}} \tag{2.5}$$

and

$$\hat{\mathcal{V}} \xrightarrow[\text{render}]{} \mathcal{V} \tag{2.6}$$

The **obscured set**, $\mathcal{O}$, is the set of objects that are not removed at the culling stage, but are not necessary for rendering, defined as

$$\mathcal{O} = (\mathcal{M} \setminus \mathcal{C}) \setminus \hat{\mathcal{V}} \tag{2.7}$$

The Z-buffering stage requires any superset $\mathcal{Z}$ of the final visible set $\hat{\mathcal{V}}$ to be processed and is thus dependent on $|\mathcal{Z}|$, the size of the visible object set, which should be minimised. Extra stages of removal which discard more objects from consideration can be inserted, attempting to reduce the set considered at the rendering stage to be close to $\hat{\mathcal{V}}$.

$$\mathcal{M} \xrightarrow[\text{cull}]{} \mathcal{V}^1 \rightarrow \mathcal{V}^2 \rightarrow \cdots \rightarrow \mathcal{V}^k \xrightarrow[\text{render}]{} \mathcal{V} \tag{2.8}$$

Each of the $\mathcal{V}^i$ is a **potentially visible set** (PVS). Obviously the costs of these stages must be carefully weighed to make them worthwhile performing. Identifying such removal operations is difficult in general, although success can be achieved in restricted applications. Some removal schemes are mentioned in Section 2.1.5.

2.1.3 Coherence

It is also important to consider the potentially visible subset as it changes over time due to movements of the viewer, or of objects in the model. A small movement in the viewpoint or the viewing direction will affect the viewing frustum and the sets of objects identified

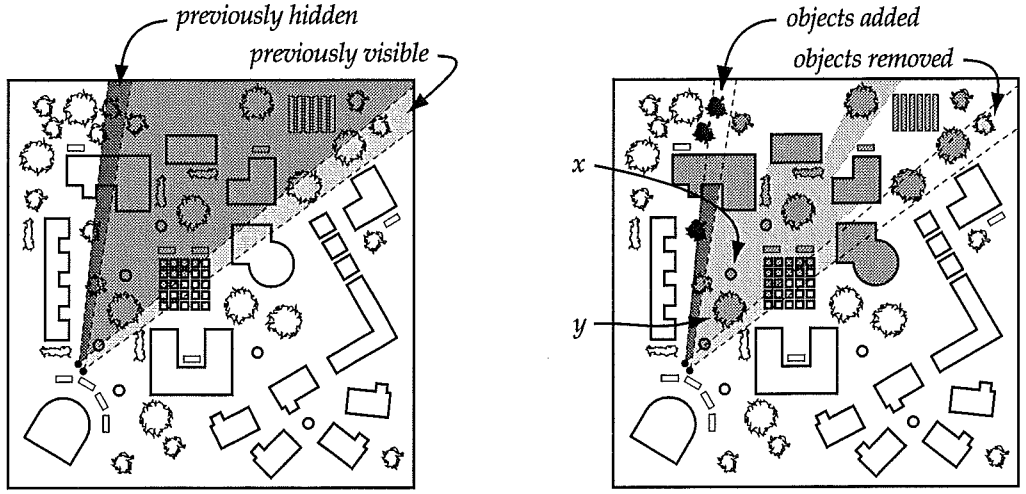


Figure 2.3: Changes in visible, culled and obscured sets due to a small viewer movement and an example of an object x that comes into view having previously been obscured by object y.

above. It can be expected that the viewing volumes for two consecutive frames are likely to overlap to a great extent, providing the viewer's movement is reasonably small (Figure 2.3 *(left)*). Due to the spatial coherence of the objects in the scene, the set $\mathcal{M} \setminus \mathcal{C}(t)$ changes slowly, and only at the boundaries.

While it might be hoped that the obscured set, $\mathcal{O}(t)$, would exhibit a high degree of coherency, this is not necessarily so (Figure 2.3 *(right)*). An example of a large jump is walking around the corner of a building, suddenly bringing into view a whole new area not previously visible.

Dynamic objects alter their position in the model as time progresses, and these too can cause changes in $\mathcal{V}(t)$ over time. Since such movement should be spatially coherent, the effect on $\mathcal{C}(t)$ is smooth and has a similar effect on $\mathcal{O}(t)$ to when the viewer moves, although since dynamic objects usually form only a small subset of the model, this effect will be smaller.

In most cases, gains in efficiency should be possible by using the coherence of these sets over time to aid in their identification at a given instant — this is particularly true for the set of culled objects which is highly coherent.

2.1.4 Calculating Potentially Visible Sets

The identification of a PVS is aided by visibility coherence — two objects that are spatially near each to other are likely to have the same visibility status. Therefore it is natural to

introduce a form of **grouping** with the aim of classifying large numbers of objects at the same time, rather than processing each individually in turn — by performing queries on the group, all its members are classified at once. The group visibility query is performed on a **bounding volume**. The bounding volume must contain all the objects in the group and, in order to be efficient, it must be much faster to test the visibility of the bounding volume, than to test all the objects in the group individually. For this reason, box-shaped or spherical bounding volumes are predominantly used, although tighter shapes have been proposed [**Kay86**] (Figure 2.4).

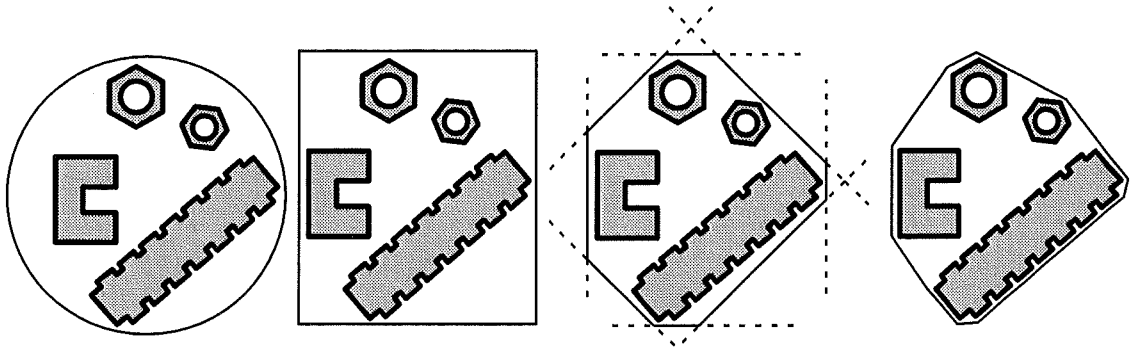


Figure 2.4: A set of objects with several possible two-dimensional bounding volumes.

For n objects split into groups of average size s, total tests can be reduced from n to n/s, by basing object status on that of the group to which it belongs. The physically larger the bounding volume of a group, the more likely it is that some part of it will be identified as being potentially visible, and will thus mark all children as visible also. Grouping based on spatial proximity will ensure efficiency in this respect.

The natural extension of this is to a **hierarchy** of groups making savings at all levels. Typically, for searching operations on a number of objects, there is a reduction from $O(n)$ to $O(\log_s n)$ searching costs when introducing a good hierarchy. By testing for visibility at a high level in the hierarchy, it may be possible to classify all objects that are below that point as invisible. This will remove large portions of the database at once, leaving the process to be applied recursively to any remaining visible branches (Figure 2.5).

Two commonly used systems are bounding volume hierarchies and spatial subdivision hierarchies :–

Bounding volume hierarchies [**Rubin80**] store objects at the leaves of a tree, and store explicit bounding volumes at internal nodes (Figure 2.6). Bounding volumes may also be stored at the leaves if objects are complex enough to merit further checking before final processing.

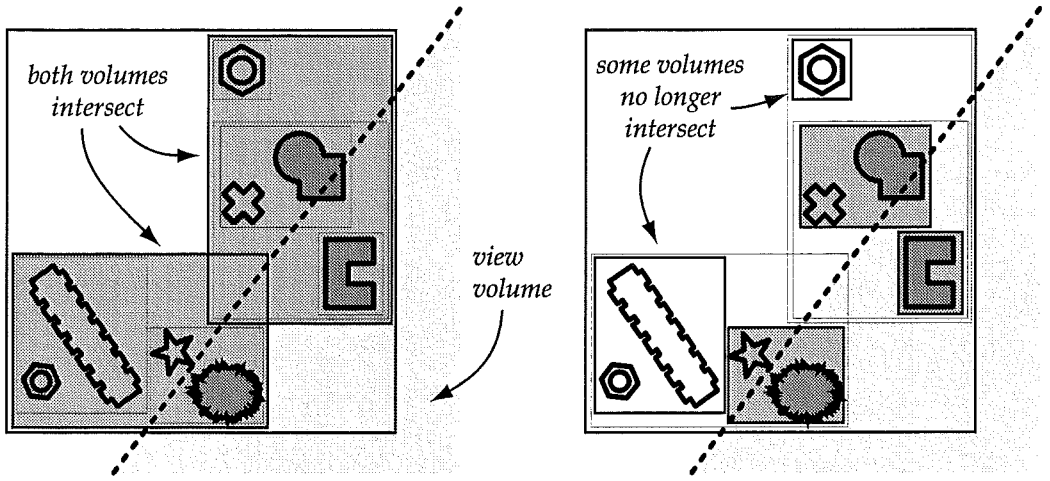


Figure 2.5: The culling operation for grouped objects showing bounding volume testing at different levels in a hierarchy.

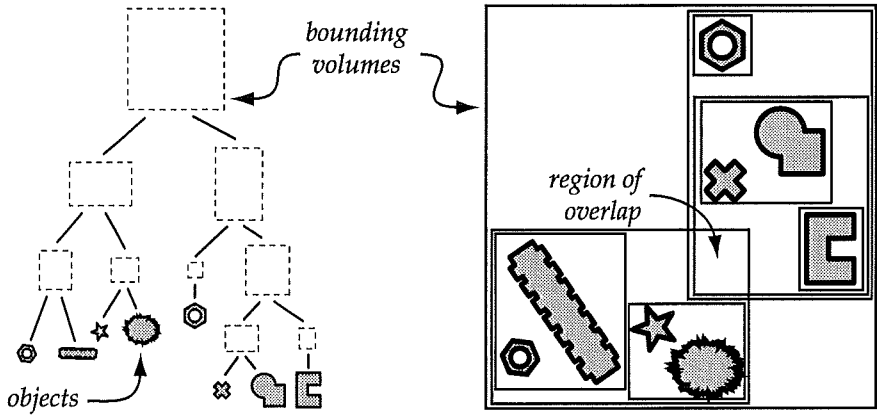


Figure 2.6: A hierarchy of groups of objects shown as a tree *(left)* and in object space *(right)* used to increase the speed of visibility classification.

Spatial subdivision hierarchies (in particular octree subdivisions [**Glassner84**]), do not have to store the bounding volumes explicitly , since the space that the objects occupy is subdivided in a known manner, hence a bounding volume can be constructed rapidly when it is needed (Figure 2.7). Objects that cross cell boundaries may either be subdivided, stored in all the cells they cross, or pushed up to lie in the smallest cell that will contain them. Other schemes include uniform subdivision [**Fujimoto86**], and the binary space partitioning (BSP) tree [**Fuchs80**].

Spatial subdivision imposes the hierarchy on the model, whereas a bounding volume hierarchy can be constructed on top of any existing hierarchy. Of the many possible hierarchies,

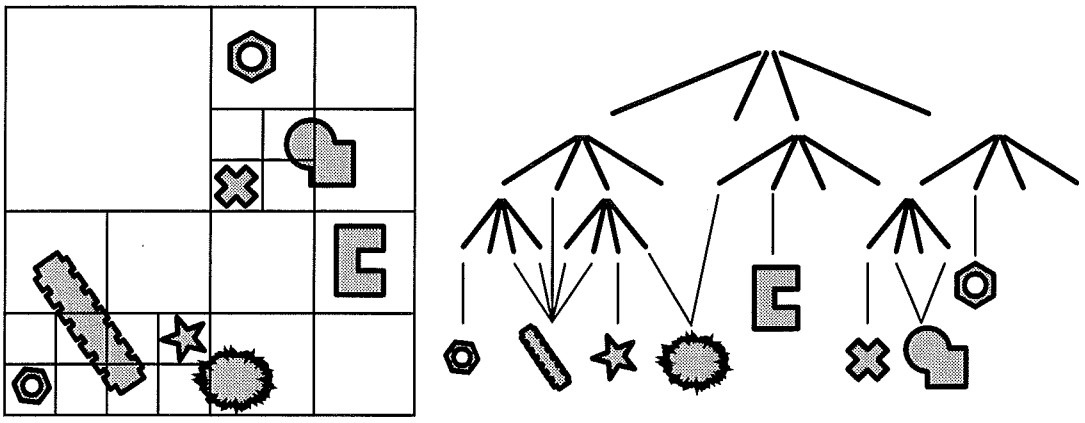


Figure 2.7: A spatial subdivision hierarchy *(left)* together with the hierarchical relationship between the objects *(right)*.

some will be much more efficient than others. Criteria for describing the efficiency of one hierarchy over another and strategies for the automatic generation of such hierarchies have been examined in **[Goldsmith87]**, **[Charney90]** and **[Subramanian91]**.

2.1.5 Visibility in Complex Scenes

As the complexity of the model increases, so will the size of the sets considered. In what proportion this occurs will depend on the data, but two extreme scenarios can be considered, namely an increase in extent of the model but with the same detail, and an increase in detail alone. Figure 2.8 illustrates these two situations schematically, and in particular the effect on the visible and obscured sets, $\hat{\mathcal{V}}$ and $\mathcal{O}$.

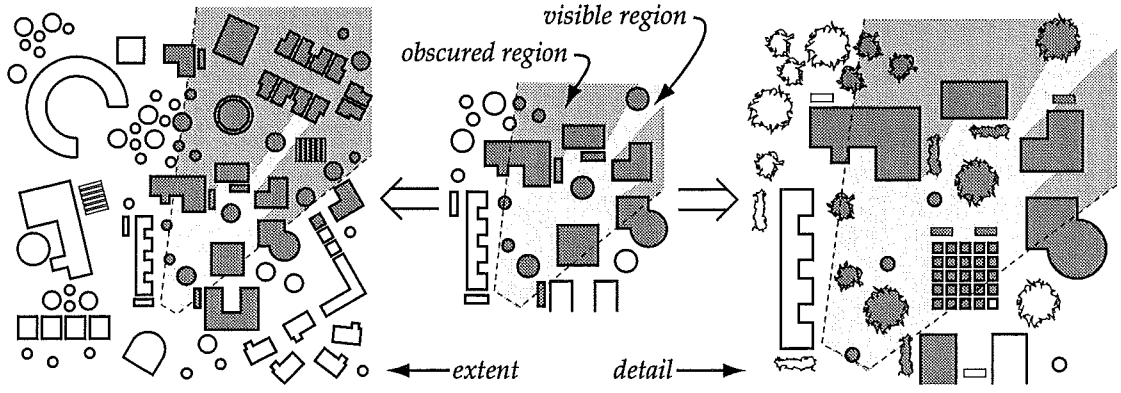


Figure 2.8: Two example situations for additional complexity — increasing the extent of the model *(left)* or the detail *(right)*.

In the larger extent case, $\hat{\mathcal{V}}$ remains roughly the same size, while $\mathcal{O}$ increases, keeping the relative proportion of $\mathcal{O} \cup \hat{\mathcal{V}}$ to $\mathcal{M}$ the same. In the additional detail case, both $\mathcal{O}$ *and* $\hat{\mathcal{V}}$ grow at the same rate. It is expected that most complexity increases will fall somewhere between these two, but will almost certainly increase the size of $\mathcal{O}$. In all situations, culling will continue to remove the same fraction of the model, while the means of identifying objects in $\mathcal{O}$ for removal will be increasingly important.

When using complex databases, it is important therefore to employ schemes that rapidly identify obscured objects. Both Z-buffering and ray-tracing permit a few immediate modifications that attempt to do this. Since a Z-buffer algorithm resolves visibility for each object with respect to the previous objects processed, some computation can be saved if objects are processed in a front-to-back ordering. While all objects still have to be scan-converted in order to establish whether they are visible, this ordering will ensure that objects are not actually *drawn* into the frame buffer unnecessarily, thereby avoiding, for example, lighting or texturing costs. The sorting step is potentially expensive but, if a spatial hierarchy is employed, can be performed roughly without prior knowledge of the entire model. This proceeds, at each branching point, by sorting the children of this branch before traversal — a step which is further aided by a regular spatial subdivision scheme.

The information stored in the Z-buffer can also be used since it holds a maximum distance at which an object may be visible under each pixel. The hierarchical Z-buffer proposed in **[Greene93]** **[Greene94]** makes use of this, together with a visibility hierarchy, to cull whole branches at a time. For performing these tests rapidly, the Z-buffer is stored as a **Z-pyramid** — a quadtree with maximum depths stored at the internal nodes. The costs of performing these tests for all objects can be significant, so the proposal includes a preliminary step that uses viewpoint coherence to calculate initial values, re-rendering the set $\hat{\mathcal{V}}(t-1)$ before constructing the Z-pyramid and using this to access the remaining objects in $(\mathcal{M} \setminus \hat{\mathcal{C}}(t)) \setminus \hat{\mathcal{V}}(t-1)$.

When tracing a ray through a hierarchy, the culling operation removes all objects except those whose bounding volumes intersect the ray. However, the bounding volume information may also be used to guide the search for visible objects. The further away the bounding volume from the eyepoint (or origin of the ray) the more likely it is to be obscured, so processing the branches in order from near to far along the ray is likely to mean that many branches will never be examined. If the intersections of the bounding volumes for each branch do not overlap, the first object intersection must be visible and all the others obscured (Figure 2.9). When these ranges overlap all bounding volumes whose intersections include this first intersection point must also be searched. Spatial subdivision schemes are particularly effective for this, although the drawback of having to deal with objects crossing boundaries is the resulting penalty. Moreover, the front-to-back path of a ray through such a hierarchy can be determined without requiring any testing of ray intersections with bounding volumes, due to the regular, cell-like nature of schemes such as octrees **[Fujimoto86]** **[Charney90]**.

Some environments are particularly amenable to obscured set identification. An architec-

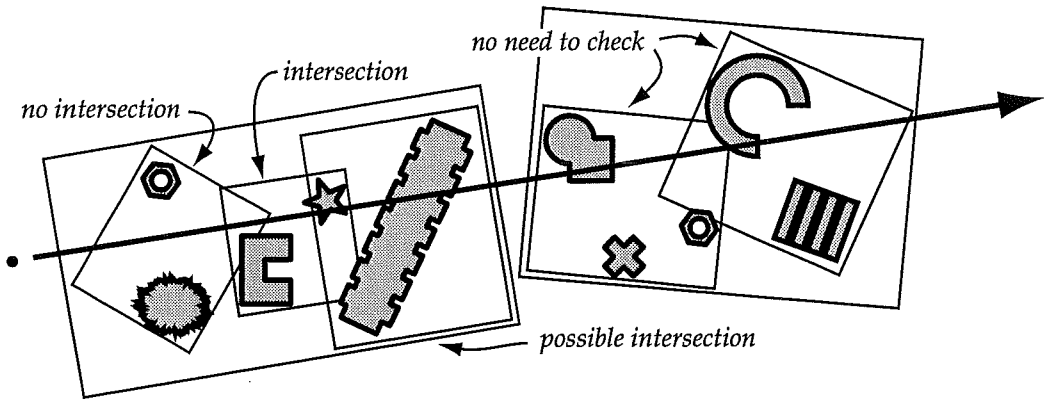


Figure 2.9: Bounding volume and object intersections along a ray, indicating which calculations are required during a left-to-right traversal.

tural model walk-through system that could be used when inside the buildings in the city model is described in **[Funkhouser93]**. Visibility from a certain viewpoint is severely restricted in such environments since the walls of rooms form barriers with only small regions that can be seen through (such as open doors and windows). Visibility pre-processing **[Teller91]** **[Luebke96]**, which will be described more fully in Chapter 6, pre-computes visibility information in order to increase the culling potential. This enables a front-to-back traversal of rooms that can be rapidly truncated if the rooms or objects in them are not visible through any visible doors or windows.

2.2 Detail

Even with a relatively efficient method for removing culled and obscured objects from consideration, there may still be too many objects. If all the complexity arises through detail rather than extent as discussed in the previous section, then however effective the visibility culling procedures, the detail will remain in the visible set $\mathcal{V}$. The detail-complexity and extent-complexity extremes used as examples in the previous section essentially only differ by the scale at which they are viewed. This section describes how detail arises and introduces a means whereby it too might be managed.

2.2.1 Scale and Complexity

In the last chapter, the need for detail at a variety of scales was introduced to allow the viewer freedom of choice of viewpoint. The practical effect of allowing this freedom to examine objects in detail is the likelihood that there will also be viewing situations where that detail is *not* being examined closely and manifests itself as unwanted complexity. For in-

stance, a tree might appear in the distance covering a small fraction of the image, or in another image, one of its leaves might fill the entire image. If there is enough detail present to satisfy the viewer when looking at a leaf, there will be far too much when the tree is in the distance. For non-realistic viewing situations, where images can be generated from the scale of planets down to that of atoms or beyond, the problems are further compounded. It is not difficult to construct a model that contains an infinitely detailed representation of an object [**Kajiya83**]. The fractal object in Figure 2.10 is an example of a visualisation of a Sierpinski Tetrahedron [**Hart91**] which is infinitely detailed.

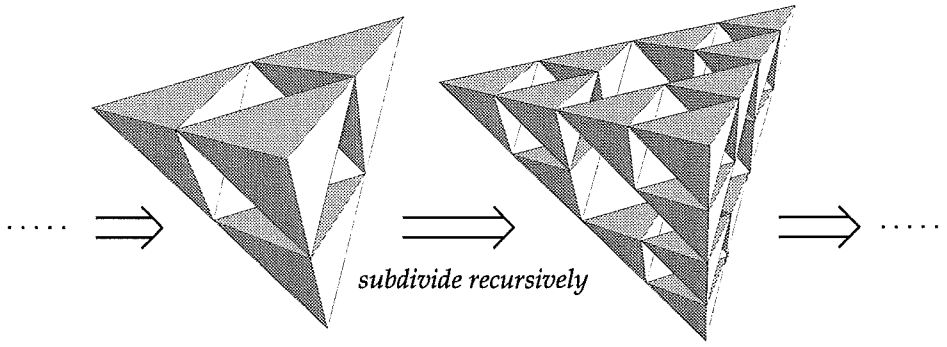


Figure 2.10: A representation of the Sierpinski Tetrahedron formed by recursively removing the centre from a tetrahedron.

Even when the scale of detail in the model is limited, the size that an object appears on the image can still vary greatly when a perspective projection is used. As the viewer retreats further away from the object, its image shrinks due to perspective foreshortening — the result appears as a smaller object with a higher density of detail (Figure 2.11). Since an object is scaled inversely by the distance from the viewpoint, the effect of an object at twice the distance is comparable to an object with twice the detail. Thus a high density of detail on the image — **image-detail** — can be introduced by either the modeller *or* the viewer.

Visibility is crucial in restricting detail — the more distant an object is, the higher the image-detail, but the more likely it is that it will be obscured from the viewpoint. However, while this is the case in some situations such as a building interior, in general, this does not remove *all* image-detail. For instance, looking out through a window, or along a street outside, could immediately give rise to a significant amount of image-detail appearing.

2.2.2 Dealing with Image-Detail

With a potentially huge amount of image-detail, it would seem that the rendering engine is still at the mercy of the model creator and viewer. To progress further, it necessary to reduce the amount of detail to a manageable level. In the first section of this chapter, the size of the problem was reduced by observing that not all objects contribute to the final result. This can

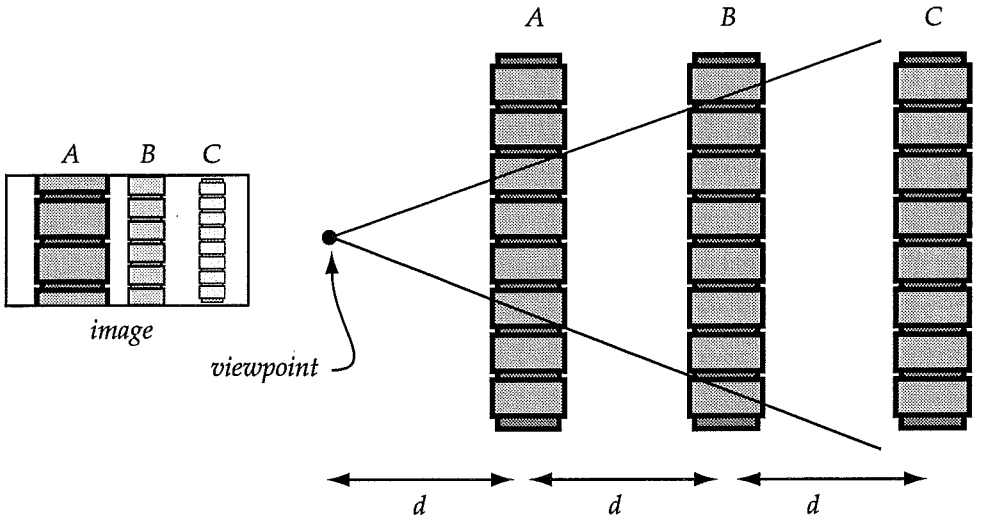


Figure 2.11: Three objects at regular intervals from the viewpoint project onto the image plane with different sizes, and hence produce different densities of detail on the image.

be amended in the light of this section to suggest that not all objects contribute *equally* to the final result, and should be treated accordingly.

As a consequence, features that appear very small in the final image need not be treated as accurately as those that make a significant impact on the image. In other words, the quality of the image should be uniform across its surface. This, however, will map back to *different* accuracies for the objects, depending primarily on their distance from the viewpoint.

Having determined that image quality should control how objects are dealt with, it is necessary to decide how this can be applied to the model. It might seem natural to suggest that details are omitted if they are too small, resulting in a method analogous to the visibility culling techniques:

$$\mathcal{M} \xrightarrow[\text{cull}]{} \mathcal{M} \setminus \mathcal{C} \xrightarrow[\text{detail cull}]{} \mathcal{D} \xrightarrow[\text{render}]{} \mathcal{V} \tag{2.9}$$

This, however, can produce disturbing effects. Consider the tree model when it is only a few pixels high on the image. Although the effect of each leaf — the detail — is minimal, their combined effect can change the appearance of the tree from summer to winter. Removing the bricks from a brick wall would leave nothing at all — almost certainly an unwanted effect.

Details therefore cannot just be removed from $\mathcal{M}$, without having something which can be used in their place. In the rest of this dissertation, these objects will be called **approximation objects** or just **approximations** — simpler objects that can be used instead of a more detailed representation in order to reduce the processing cost. In that respect it is similar to

the use of bounding volumes in visibility calculations — Equation 2.9 is now modified to

$$\mathcal{M} \xrightarrow[\text{cull}]{} \mathcal{M} \setminus \mathcal{C} \xrightarrow[\text{detail choice}]{} \mathcal{D} \xrightarrow[\text{render}]{} \mathcal{V} \tag{2.10}$$

2.2.3 Levels of Detail and Approximation Hierarchies

The use of models with multiple **levels of detail** (LOD) has recently become more common, with several systems offering support. A typical example is for a polygon mesh model of an object where several different versions of the same object are available with differing numbers of polygons (Figure 2.12). Instead of deciding which of these will be inserted in the model, the designer includes all of them, with some hints as to when each should be used. For instance, IRIS Performer has a LOD object where different children are drawn depending on the distance of the object from the viewpoint [**Rohlf94**].

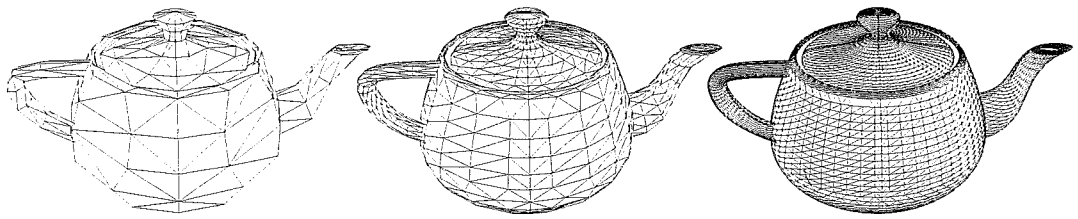


Figure 2.12: Example of a polygon mesh representation of an object modelled at different levels of detail.

Using an LOD object to switch between different quality representations of each object enables the renderer to choose the least detailed, and hopefully fastest, representation that is sufficient in the particular situation. Although such a use will result in increased performance, it is rather similar to having a bounding volume visibility scheme where bounding volumes are only kept for each object, and not in a hierarchy. To illustrate this, while a tree might be stored with three or four levels of detail, when a forest is drawn from a distance, each tree will still have to be drawn separately at its lowest level of detail. As the viewer moves away to view an entire landscape from the air, this requires too much detail since the number of objects in view is increasing even though individually they are drawn more simply.

Another difficulty is that if an object is large, and viewed obliquely (Figure 2.13), then the parts in the foreground need to be of significantly higher quality than those in the distance. This will occur when the size of the object is much greater than the scale of the detail represented on it. If quality can only be set for the entire object, then compromises will be necessary. As was true in the visibility case, the conclusion to be drawn is that approximation should be performed in a hierarchical manner.

By replacing groups of objects, not only should the polygon count (or whatever is the determining complexity factor) be reduced, but the total complexity of the image should also

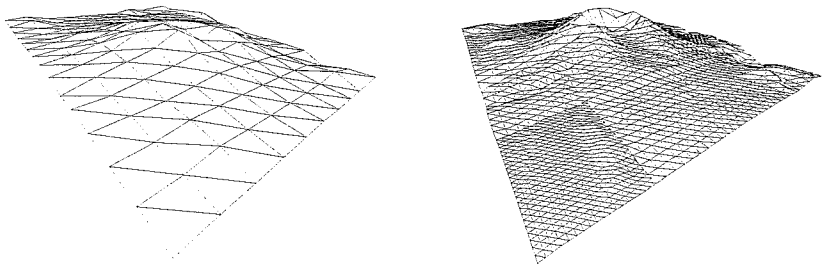


Figure 2.13: Large objects viewed obliquely such as the landscape *(left)* require a variation in level of detail *(right)*.

drop, since there will be less self-occlusion. This is of particular benefit when the detail being replaced is so small that it will not be missed.

2.2.4 Approximation at the Pixel Level

Most computer graphics rendering techniques are based around point sampling. Ray-tracing is the obvious example of this, but the same assumptions are made within most polygon scan-conversion routines. These methods all use a subdivision of the image plane, usually at pixel centres, as the basis for the location of sample points. Some methods are capable of sampling the model at different resolutions, such as the adaptive subdivision method used within a ray-tracing system. Many radiosity methods employ sampling over the objects in the environment rather than over the image, tracing rays between pairs of points on source and receiver surfaces to establish visibility [**Wallace89**]. While having some of the features of object space sampling, this is still point-based.

Point samples are used to approximate a weighted area sampling which would be an ideal, but impractical, solution. One improvement, most familiar from two-dimensional image manipulation, is to perform **pre-filtering** before sampling [**Foley90**]. If for instance a local averaging filter is applied to a signal (Figure 2.14) and the result sampled, then the samples will be of the local average, thus achieving the required area averaging at the cost of a large amount of pre-processing (the pre-filtering step). Unfortunately, the filtering is dependent on the sample spacing, and if this is not known in advance, it has to be done at a range of resolutions. A similar approach for texture mapping, MIP-mapping, is now common practice, where a sequence of textures at lower resolutions is produced from the original high-resolution texture by area averaging [**Williams83**]. At display time, the most appropriate to the sampling resolution is chosen (or a linear blend between the two nearest resolutions is sometimes used). Hierarchical radiosity algorithms [**Hanrahan91**] use a similar approach to store radiosity solutions over the surfaces of the model, so that suitable resolutions can be used depending on the level of the sampling. Each of these examples uses the detail information to generate the averaged values in advance.

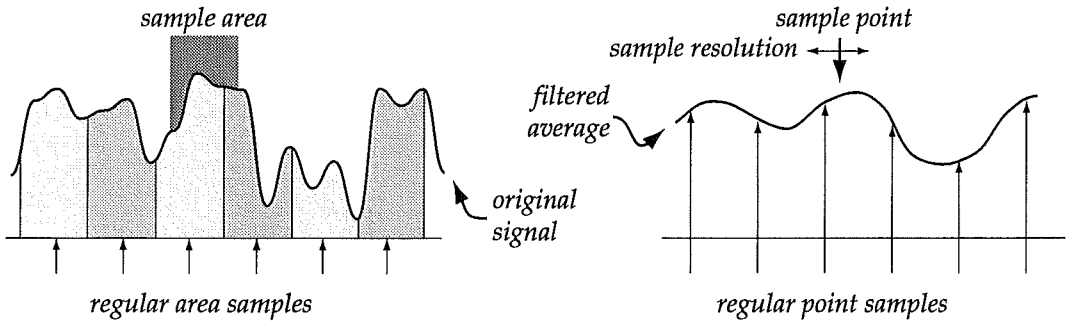


Figure 2.14: Pre-filtering a signal before point sampling to simulate area sampling. The bands indicate the sampling and averaging resolution.

The replacement of detailed objects by approximations can also be viewed as a pre-filtering process. If the approximation is formed with this in mind, then when an approximated object is point sampled, the effect will be similar to an area sampling of the original, and thus "pre-filters" the model, reducing aliasing effects. To work well, the level of smoothing and the sampling rate need to be closely matched. Certainly, considering the tree model which covers only a few pixels, an adequate sampling rate would have to be high enough to sample each leaf several times and would require a large number of samples. Using a good approximation should require only a few samples in those pixels to obtain a better result (Figure 2.15). It should be noted that this pre-filtering reduces aliasing of detail in the body of an object, but *not* the detail arising due to the boundary between two objects.

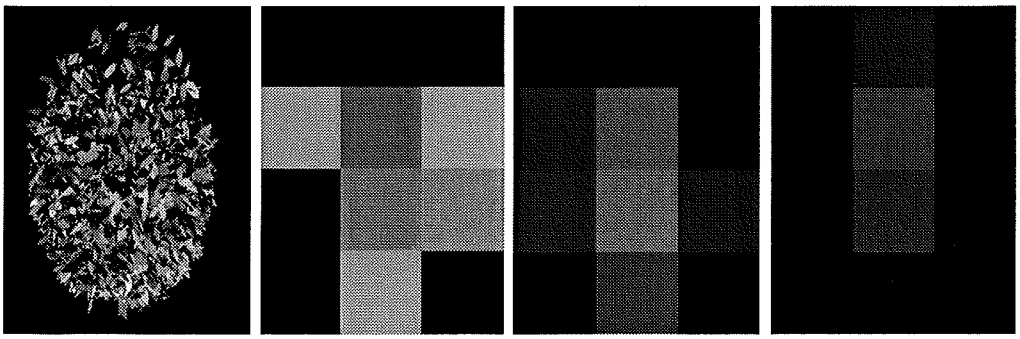


Figure 2.15: A simple tree containing 5000 leaves *(left)* and three low resolution images — the first has one sample per pixel, the central one has 64 samples per pixel, and the last is again one sample per pixel, but this time of a simple approximating object.

2.3 Problem Integration

The previous section provided the motivation for including approximations in the model. Together with visibility considerations, this will enable the application to choose a subset of the model with which to work at any given time. Furthermore, the size of the subset can be controlled in principle by choosing the quality at which objects are to be included and excluded from this subset. This section provides a structure within which such decisions can be made in the form of an **approximation hierarchy**, but starts by looking at how the concept of approximation can be extended to embrace some of the other aims stated at the end of the first chapter.

2.3.1 Modelling and Approximation

Approximation already plays an important role when a model is being constructed. The designer chooses an abstract representation of the object to be modelled, and constructs a version of this in the computer. Almost all models built in this way are simply approximations to some more ideal conception of the object. When modelling a chair as a number of polygons, for example, the modeller will ignore the detailed curvature of the surface, or perhaps some of the roughness of the seat covering.

There are some objects, such as spheres and other mathematically defined primitives, which can be represented accurately in the computer, but even in these cases, it is not always the mathematical representation that can be used; a sphere might have to be split into a large number of planar polygons for rendering in hardware.

As mentioned in Chapter 1, there may be several different representations created for the same object; a sphere can be represented either directly for use with a ray tracing system, or as a polygon mesh for a Z-buffer system, or perhaps as curved surface patches for a CAD modeller (Figure 2.16). Similarly, a chair could be constructed either from boxes and cylinders, perhaps using CSG, or modelled as a polygon mesh, or even as volume data. As is the case for the sphere, these different approximations are of different quality as well as of different types.

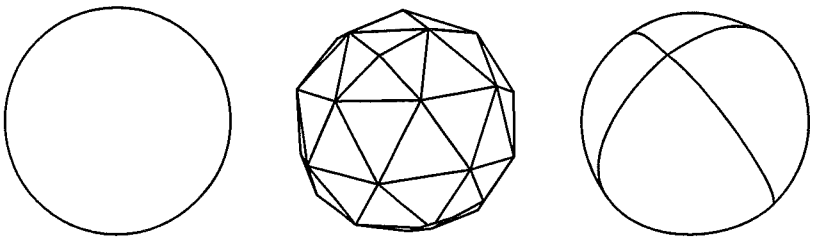


Figure 2.16: Some different representations of a sphere, as a mathematical primitive, as a polygon mesh, or as a set of surface patches.

Just as representations are often *approximations* of different quality, the approximations, introduced in Section 2.2, are essentially just different *representations* of the same object.

2.3.2 Approximation for Type and Format Conversion

In order to support a wide range of *types* of data, applications will need to have the ability to convert representations of a type they cannot handle to a representation of one they can. It will not always be possible to do this without a loss of quality, as can be seen when polygonalising the sphere. Just as different representations are equivalent to different approximations, the process of type conversion is really just a matter of approximation — representations of one type being replaced by representations of another.

If an application requires data of a particular type, perhaps consisting only of triangles, then it should look for a representation of that type. If a suitable representation has been calculated and stored in advance, then this can be used. If none is available, then an approximation routine can be called to convert another representation to one of the desired type. The approximation could be either the result of human effort at the modelling stage, or the output from an automatic or human-guided approximation algorithm. If it is to be done at run-time, then an automatic process is necessary. For many type conversions, this might be quite straightforward — by converting polygons to triangles, or geometric primitives such as boxes or spheres to polygons. Others are more complicated, such as the marching cubes algorithm for creating iso-surface representations from volume data [**Lorensen87**]. During an approximation process, it is likely that either the size of the representation will increase, or the quality will decrease, or both may occur.

While this might encourage larger amounts of data and slower access times due to approximation or conversion methods having to be run, it is the only alternative to standardisation. Although it might suit many current rendering systems to simply convert all surfaces into polygon meshes, this is wasting valuable information, and will penalise systems in the future that could run more efficiently with another representation. Polygonalisation restricts the types of objects that can be modelled, and in many cases will greatly increase the storage size necessary. In order to fulfil the demands for detail, these polygon meshes could become huge, and would require algorithms to reduce them to simpler forms in order to facilitate their display on less powerful hardware. While such approaches should be considered, they largely contradict the aims set out in Chapter 1.

In order to give transparent access to the sort of model described in Section 1.5.3, the final issue that must be examined is how to deal with a representation of an object which is of the right type but is stored in a different *format* — having a list of triangles is no use unless it can be read into the system.

Format conversion is essentially a simple case of type conversion, where the stored data alters but the actual representation it describes does not. This can be achieved in a similar framework to that described above by keeping the quality fixed. It should be noted,

however, that in most practical cases formats are not identical and some degree of approximation will occur. This may not affect the physical structure of the representation as seen by the viewer, but might alter the structural way in which the data is stored, for example by converting polygon meshes to polygon lists.

2.3.3 Approximation Hierarchies

A new framework, the **approximation hierarchy**, is introduced in this dissertation to structure models. An approximation hierarchy is a hierarchy including **approximation nodes**, which can be imagined as nodes where a choice must be made. In order to use the information present in an approximation hierarchy, it is sufficient to choose only *one* path out of each approximation node. The result of doing this is to traverse a **representative subtree**.

More formally, a hierarchy is formed from **nodes**, each having any number of children. These nodes are collected together into a directed network as shown in Figure 2.17, where the directed arcs between nodes indicate the relation "is a child of". A rooted hierarchy is a hierarchy together with a unique **root node**, where all nodes are descendants of the root node in the sense that they can be reached by following a series of child relationships from the root.

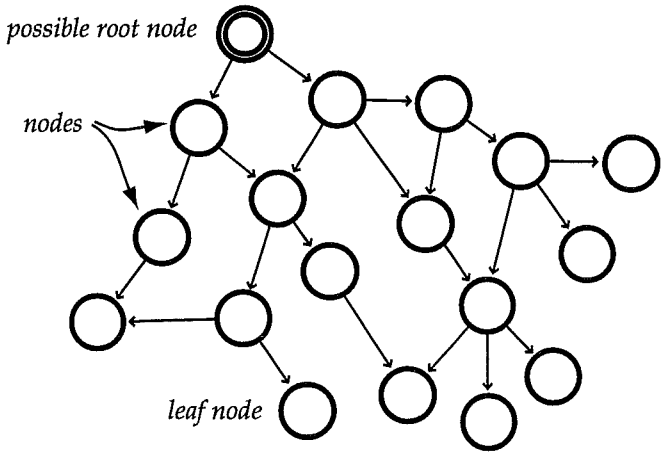


Figure 2.17: A hierarchy as a connected network of nodes with a possible root position marked.

An **object hierarchy** is used to describe a representation of an object. Geometrical (and other) information is stored in the nodes of the hierarchy. A **traversal** of an object hierarchy visits all nodes in the tree formed by following paths from the root, and it may be helpful to consider the children of nodes to be ordered during this. In order to guarantee that this traversal terminates, the restriction has to be made that an object hierarchy contains no backward loops (making it a directed acyclic graph or DAG), although this restriction can be relaxed in approximation hierarchies. This does not restrict some nodes being traversed

several times where they are the children of more than one node, just that no node should have a child that is on the path from the root to that node (Figure 2.18).

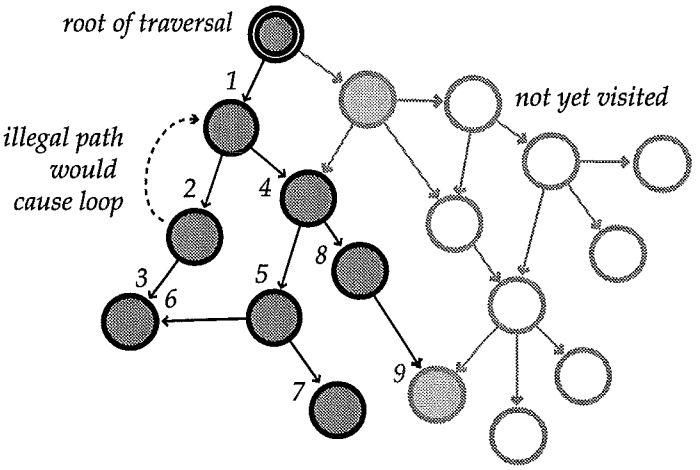


Figure 2.18: A depth-first traversal of an object hierarchy pictured halfway through. Numbers indicate the order of traversal and the black nodes are those that have been visited.

An **approximation hierarchy** is an extension of an object hierarchy. The extra approximation nodes behave rather differently under traversal. Instead of traversing each child in turn, only one child is selected for traversal, so that the nodes acts like a switch, exchanging one child graph for another (Figure 2.19). An approximation hierarchy allows arbitrary numbers of representations of an object and is, as such, an expansion of the hierarchical objects first described in **[Clark76]** and later **[Crow82]** **[Thompson91]** and **[Maciel94]** which only use a single alternative representation.

A **representative subgraph** is a subset of an approximation hierarchy where the children of approximation nodes are pruned, provided that at least one child remains for each approximation node. If the approximation hierarchy is a valid representation of an object, then any representative subgraph is also a valid representation. A minimal such subgraph would leave exactly one child at each approximation node.

An **abstract object** is in essence an approximation hierarchy (together with any additional information that needs to be stored) and different representations of the object are described by representative subgraphs.

2.3.4 Quality

In order to choose between representations, particularly for deciding how detailed a representation is necessary, it is important to be able to make a comparison. This process can be described in terms of making decisions based on the values of sets of **quality variables**.

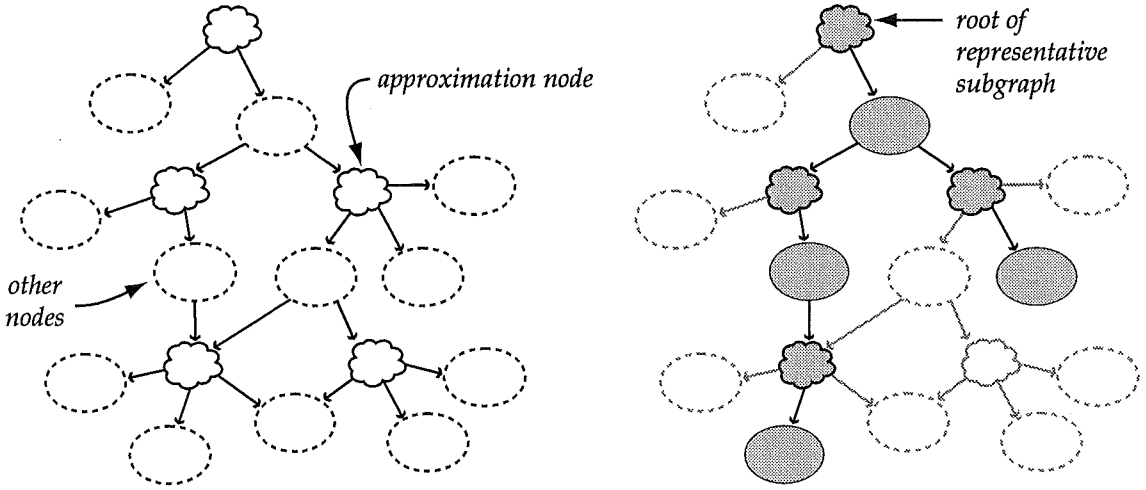


Figure 2.19: An approximation hierarchy (where non-approximation nodes are shown grouped) and a representative subgraph where exactly one child relationship is followed for each approximation node in the subgraph.

While it would be notationally neat to be able to prescribe a single quality variable that is universally applicable, this is not a realistic proposition since quality can unfortunately depend on the context in which it is to be used.

Quality variables occur in two situations, firstly as a means for the user to control the application, and secondly so that the application can control itself. In most systems there are already many "quality control" variables. Setting these correctly and consistently is difficult, and while they individually allow images to be fine-tuned, in most circumstances such low-level control is not required by the user. One regrettably common usage is to turn all the settings up to maximum before the final rendering in order to get the best results, despite the fact that this might entail a huge computation for little noticeable improvement in image quality.

An ideal system would present a hierarchy of controls, each of which by default determines its values from a higher-level control but which may also be altered individually (Figure 2.20). The novice user can then be presented with a single measure of quality with which to control all aspects in a consistent manner; only if a special effect is desired need other controls be examined. Thus, if a rough model is being used for rendering, then low quality shading and lighting are also utilised, but if the user increases the quality control, then the shading and lighting *as well as* the model should become more accurate.

Internally, the application needs to select representations rapidly based on some quality variables. Typically, this is most convenient if comparisons can be made on a single quality variable, or quality **criterion**, which might be computed from several quality variables de-

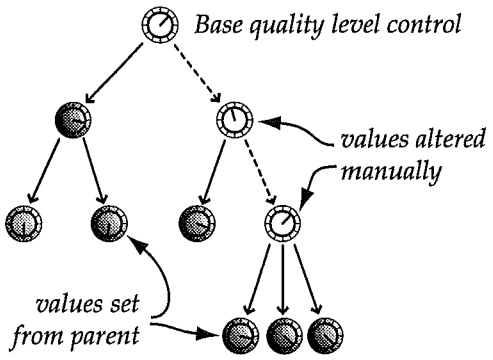


Figure 2.20: A schematic hierarchy of quality controls giving the user control at any level, without requiring unnecessary expert knowledge.

pending on the context of the application. Chapter 5 assesses some example criteria; Chapter 3 and Chapter 4 introduce other quality variables used for describing approximation quality.

2.4 Summary

The emphasis in the present chapter has been on devising a means of reducing a potentially massive problem to a level that can be realistically handled by users, and a range of hardware. This is achieved by restricting applications to dealing with only a part of the model at any one time. This section proposes a **model manager** as an effective means of providing an interface between a stand-alone model and an application.

2.4.1 The Model-Centred Approach

Two methods have been suggested for reducing the size of the model that needs to be considered at any one time, namely visibility culling and detail replacement. These are based on the principle that it is not necessary to deal with all objects at the same detail at once; a model subset is sufficient for drawing some representation of the model. Furthermore, this subset is expected to change slowly over time, due to the many forms of coherence inherent in 3D models and viewing paradigms. Within this framework, systems no longer need to load the entire model in advance but can instead maintain the currently required subset from a model which exists outside the application. The model subset, together with approximations, produces a system that can scale with the hardware it is running on in a way similar to the working set solution first proposed in [**Clark76**]. This follows the principle of enabling data to be viewed at different levels of information content, proposed in [**Read93**] as a system-wide aim for multi-resolution.

Approximations have also been used as a framework for incorporating objects of different types into a single system, allowing applications to deal with any object at some level, although perhaps only as a crude approximation. By using approximations to all the visible objects, the size of the partial model can be reduced so that it can fit within the memory and rendering limitations of the hardware. Alternatively, the size can be set so that the required rendering speed can be achieved, allowing for low-power hardware to display a simplified model in real time.

2.4.2 The Model Manager

The attraction of having the model inside the application is the ease with which it can be processed by the application. Supporting the model-centred approach, and also maintaining the subset as requirements vary, place additional burdens on the application. These tasks are in essence independent from the application itself, as they relate only to the model. The model manager forms the interface between the application and the model, and encapsulates the tasks associated with fetching the object and maintaining the model subset (Figure 2.21).

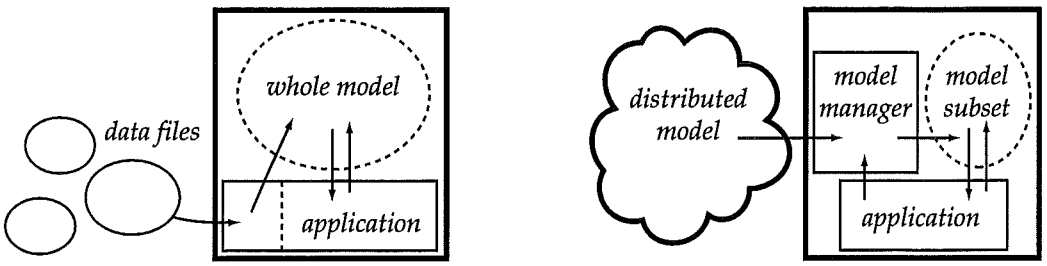


Figure 2.21: Traditional relationship between the model and the application *(left)*, and the model subset scenario with the model-manager providing the interface between the separate model and the application *(right)*.

The idea of the model manager is to present to the application, as best it can, the illusion of access to the entire model. The virtual model that the application sees has been filtered for its consumption, so that, for example, it only contains types of representations with which the application can deal. At the same time, the manager may only be maintaining a small subset of the model, and when fetching new items may have to convert representations to present the data in the required format. Furthermore, the model manager can also fulfil the other aim of presenting a uniform interface to what is probably a complex, distributed model.

2.4.3 Overview

A system that performs the tasks proposed in these first two chapters requires many different elements. The remaining chapters of this dissertation aim to examine the most important of these.

The next two chapters are devoted to examining how approximation schemes might work. Chapter 3 looks at the problems of approximation, and in particular simplification, and examines how some current algorithms can be adapted to this task. Several new approaches for performing approximation successfully and efficiently in new situations, in particular when replacing groups of objects, are described in Chapter 4.

While a model manager can isolate many issues of complex models from the graphics application, there are still some basic capabilities the application will need in order to use approximation hierarchies. Chapter 5 examines basic systems that can use approximate data, and describes developments which allow for more interesting uses of approximate models.

The model manager is the subject of Chapter 6, which looks at the design issues and decisions important in constructing a model manager for a given application.

Chapter 7 returns to the aims set out in the introduction and considers how much progress has been made in achieving them. In addition, areas for future research, which might be expected to yield improvements and further innovations, are summarised.

Chapter 3

Data Approximation

— In which the approximation process is examined and some simplification algorithms are described.

The most important mechanism described in the last chapter is approximation. By including approximations in a model, an application can choose to process only a manageable amount of data. This chapter, and the next, describe techniques for generating approximations, concentrating on algorithms for creating a new representation of an object. As well as the geometry, other aspects of the model, such as surface properties or even behavioural properties, can be considered as subjects for this operation. Most of this discussion is concerned with geometry although Section 4.4 in particular discusses surface descriptions.

While Chapter 4 looks in detail at new methods that can be applied in order to approximate complex objects, the present chapter starts by examining how approximation algorithms might work in general. The first section analyses several issues relating to approximation; in particular, it distinguishes between **simplification** and **replacement**, and discusses **multi-stage** approximation with an emphasis on using approximations in the new contexts identified in Chapter 1. The rest of the chapter is devoted to simplification methods. Several existing algorithms are re-examined in the context of generating simplifications of polygon meshes, parametrically defined surfaces, and volume data.

3.1 Approximation

The first point to consider is what constitutes a "good" approximation — which of course depends on the *context* in which the approximation occurs. Object approximations may be used for two distinct purposes:

- to be *indistinguishable* from the original,
- to convey enough information to provide a reasonable *impression* of the original.

These are usually separated by the *scale* at which the approximation is used — when only a few pixels high, an approximation may look identical to the original, but it may also need to be used at a much larger size. The former case can be discussed in terms of the differences of pixel values in the image, but the latter is as much to do with human perception as geometrical accuracy.

In the framework introduced in the previous chapter, approximations to an object are considered as different representations and vice versa. In practice, it is not the abstract object that is always the subject of approximation, but rather one particular representation which is used to generate another; the terms object and representation will be used interchangeably in this chapter and the following one.

3.1.1 Strategies

Since it is impossible to predict in advance all the uses to which approximations will be put, or what object types may or may not be in use at some future time, it is also impossible in advance to provide a comprehensive answer to the question of how to perform object approximations. For these reasons, a single-high quality algorithm is an unlikely prospect. Instead, specialised approximation techniques are more likely to be developed, each producing good approximations for a restricted set of input object types.

While being able to generate such high quality approximations is useful, it also very important to be able to generate automatically at least *some* approximation to *any* object. In particular, if a group of objects of assorted kinds is to be replaced by a single approximation, then only a more general technique is likely to succeed. Since such a method is to be applied to general input types, it is constrained to using information that can be obtained in a general framework, and therefore will only be able to provide general solutions which are unlikely to be of high quality. The inaccuracy, however, will be outweighed by the benefits that accrue from being able to approximate *anything*. This area has not been the subject of much investigation — the methods presented in Chapter 4 aim to do just this in order to use approximations with as wide a spread of model types as possible. In contrast, the present chapter is more concerned with examining *current* methods and *current* object types.

Automatic methods are important in order to minimise the burden on designers (as pointed out in *e.g.* [**Heckbert87**]). At the same time, it is essential that designers are free to guide

or even override any automatic procedures to replace an automatically generated approximation. Approximation generation itself is just one part in a three stage process, and the modeller should be able to override each stage.

- **structuring** choosing what to approximate. If a large number of objects are given to the algorithm, approximation should not always be applied to the whole set at once. Instead a hierarchical set of groupings and replacements is often the better approach.
- **approximation generation** generating an approximation for each group of objects.
- **quality evaluation** measuring how good the approximation is and deciding when it can be used by the application.

In order for object approximation to be general enough to support the separate-model approach, it must be done independently of the rendering that might be applied to it later, although there may be situations where renderer-dependent decisions can be made. Approximation, therefore, is discussed in terms of a physical operation on an object, and measurements and algorithms must be applied in object space.

3.1.2 Object Types and Approximation Paths

One familiar situation in which approximation is sometimes applied is inside a **format converter** whose task is to read a model description in one external format and write out another model description in another external format. This involves three stages (Figure 3.1) :-

- parsing the first external format to create an internal representation of the model,
- converting the objects and structures to another form,
- writing the result out in the final external format.

When the models described by the two external formats are not identical, the interesting stage is the central one, which has to convert objects of one type into objects of another.

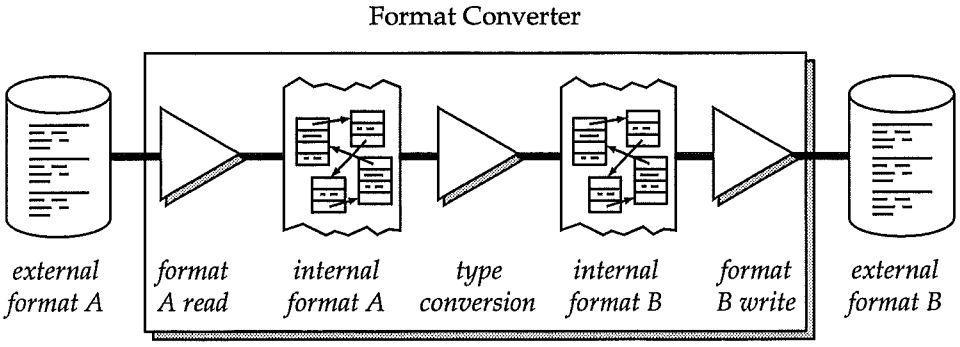


Figure 3.1: The structure of a format converter.

Currently, a number of primitive object types are in use, such as spheres, polygon meshes, boxes, particle systems, density functions, Bézier surfaces *etc.* Approximation routines can be defined that, for example, replace a box by the six polygons from which its surface is formed, or replace a Bézier surface by a polygon mesh. Figure 3.2 *(left)* shows a **type conversion graph** that describes the transformations possible with a given set of **type converters** in terms of nodes t_i representing object types, and arcs $c_{ij} = t_i \rightarrow t_j$ corresponding to conversion routines.

By applying multiple converters a larger set of transformations is possible, so that the conversion $t_{i_0} \rightarrow t_{i_1} \rightarrow \cdots \rightarrow t_{i_{q-1}} \rightarrow t_{i_q}$ can be achieved using the converters $c_{i_0 i_1}, c_{i_1 i_2}, \ldots c_{i_{q-1} i_q}$ (Figure 3.2 *(right)*).

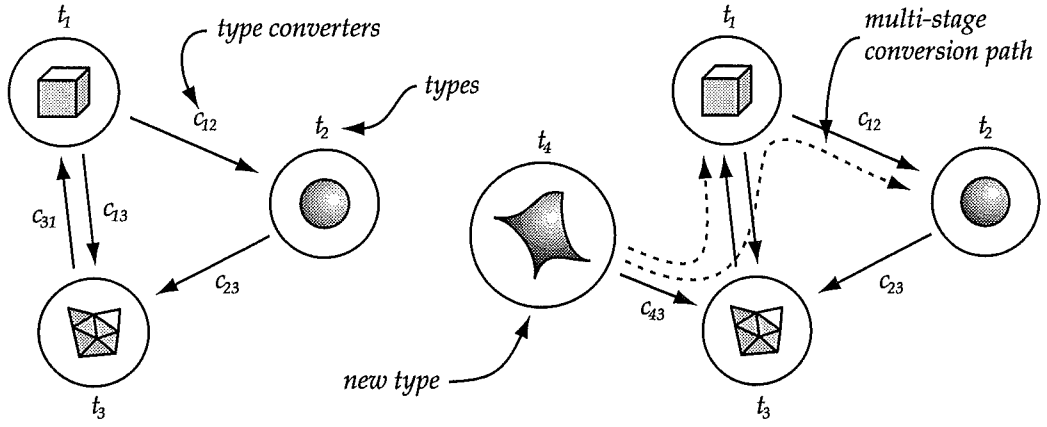


Figure 3.2: A simple type conversion graph *(left)*, together with the new graph when an additional type has been inserted *(right)* also showing multi-stage conversion paths.

If there are multiple paths from t_i to t_j, a choice can be made based upon the quality of each path. For example, converting a sphere to a cube then to a polygon mesh will produce a much lower quality result than converting directly to the polygon mesh (Figure 3.3). Approximation paths could be found by solving a simple shortest-path problem in the conversion graph. Unfortunately, the costs and quality degradation along each link are a function of the conversion method, and will probably depend also on the input data. When choosing paths through the graph, a single weight is the optimum means of describing each path, and a mapping on to such a measure is an advantage.

The distinction is made between type converters and format converters because a given type converter that, for example, converts a sphere into a polygon mesh, can be used on *any* sphere, irrespective of the format in which it has been stored. In practice, format converters tend to be more interdependent than this, since the structures used to store objects and their component parts can be quite varied. The format determines the set of types of objects that occur, and hence the type conversion routines that may be required for that format. A format conversion graph, similar to the type conversion graph above, can also be

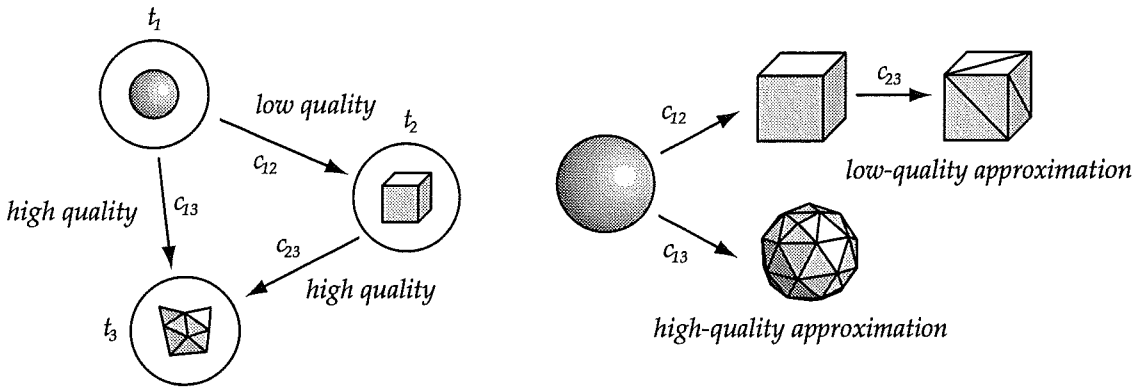


Figure 3.3: Two different paths through the type conversion graph *(left)* yield different quality conversions *(right)* for an example object.

constructed to control external format conversions (Figure 3.4). Chapter 6 describes an implementation of a system that uses format and type converters.

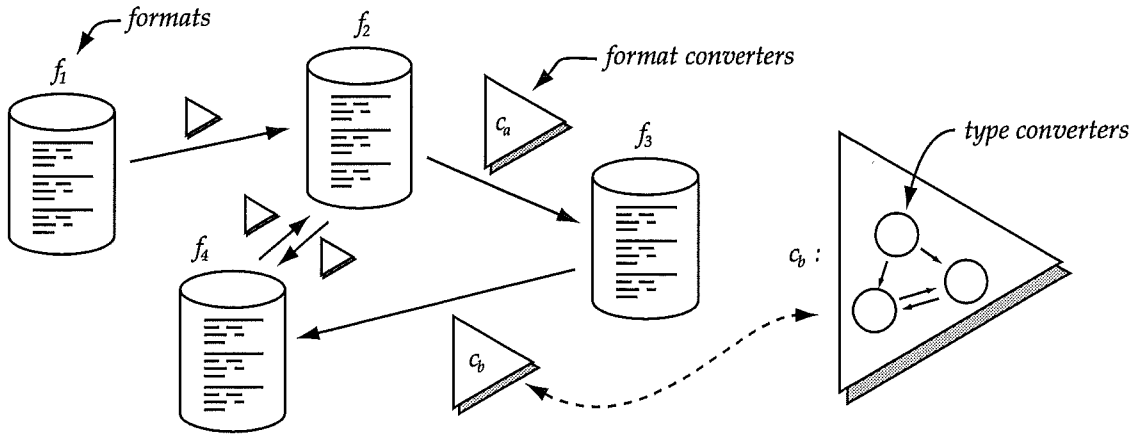


Figure 3.4: A format conversion graph.

Multiple stage approximations make it simpler to add new types of objects or file formats, since initially only a single link needs to be specified that will join the new type into the existing conversion graph (Figure 3.2 *(right)*). While higher quality conversions may eventually be required, this minimises the effort necessary to insert new types of objects.

Disconnected conversion graphs are an obvious hazard, since there will be no means of converting between a node in one half of the graph and a node in the other. By forming the transitive closure of the graph, this accessibility can be assessed, since this graph will contain single links corresponding to each possible approximation *path*. The nodes that have no incoming paths are **source** nodes, and can never be reached during an approximation

process. All nodes which do not have a path leaving them are **basic**; an application must be able to process objects of these types, since they cannot be converted to another type.

3.1.3 Simplification and Replacement

In order to aid navigation through the varied approximation schemes possible, a categorisation is proposed. Two classes of approximation strategy can be distinguished — **simplification** and **replacement**.

Simplification takes an object description and modifies the description to reduce its complexity, keeping the type of the object the same. Bearing in mind that types of objects may be either **simple**, such as a primitive like a sphere, or **compound**, such as a group object, like a list or an array, examples of simplification include (Figure 3.5)

- reducing the number of triangles in a polygon mesh,
- filtering a volume density distribution to remove high frequency components,
- removing some small objects from a list of objects.

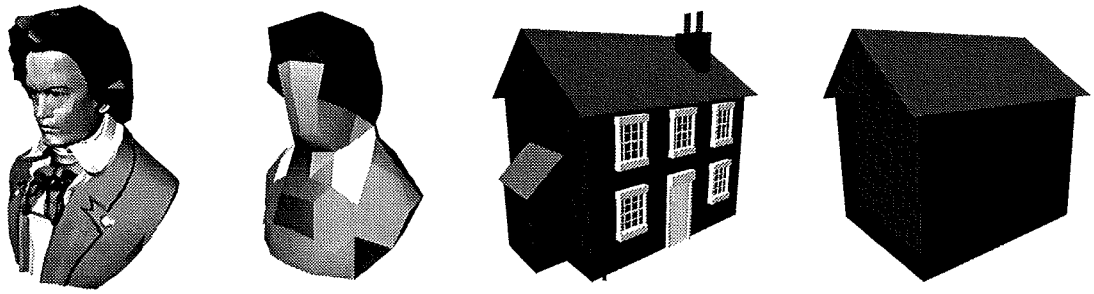


Figure 3.5: Two objects and possible simplifications formed by reducing the number of polygons in a mesh *(left)* and removing small details from a list *(right)*.

Replacement on the other hand takes the original object and generates a *new* object that approximates the first. Examples of replacement include (Figure 3.6)

- replacing a group of objects by a sphere,
- texture-mapping a simpler shape with a picture of the original objects,
- approximating a curved surface by a polygon mesh.

Only some types of object will admit simplification — there must be an aspect of their specification that can be reduced; for example, the number of polygons in a polygon mesh, the degree of a curved surface, or the number of items in a list. *All* object types are possible sources for replacement, and similarly all object types are permissible as replacements, although it is unlikely that some would ever be used for practical reasons such as efficiency. Simplification techniques by definition are specific solutions for certain object types only,

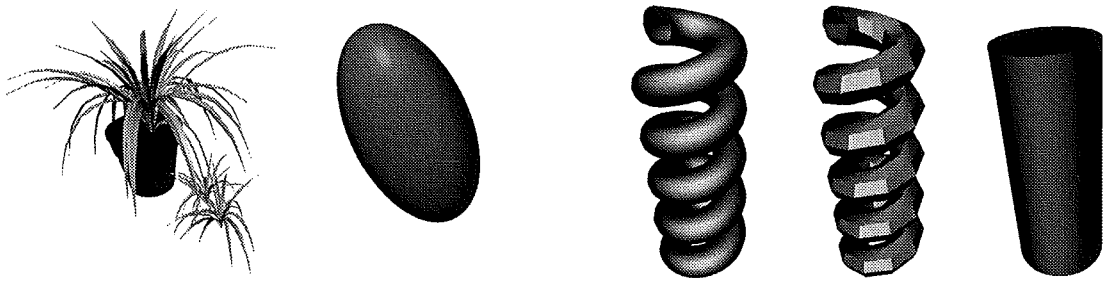


Figure 3.6: An example of replacing a complex group of parts by a simpler shape *(left)* and replacing a curved surface *(right)* by a polygon mesh or a different primitive.

whereas replacement techniques allow more general solutions. Some simplification methods on the other hand have the potential for generating objects at a nearly continuous range of quality; replacement is rarely as controllable.

3.1.4 Attribute Matching

The idea of an approximation to an object is that it is similar to the original. **Attribute matching** is introduced as a means of formalising this. It attempts to measure similarity in terms of a set of **attributes** or properties that can be calculated for the original and the replacement. In fact these attribute values can be used to guide the construction of the approximation so that it has the required properties. A practical method using this approach for object replacement is given in Chapter 4.

If object A described by a set of properties $P_i()$ with values

$$P_A = (P_1(A), \ldots P_n(A)) \tag{3.1}$$

then an approximating object B should be constructed so that,

$$P_i(A) = P_i(B) \text{ for } i = 1 \ldots n\,. \tag{3.2}$$

If the set of properties is too restrictive then the error between the sets of attribute values should be minimised. An error metric can be defined as

$$|E| = |(E_1(A,B), \ldots E_n(A,B))| \tag{3.3}$$

where

$$E_i(A,B) = |P_i(A) - P_i(B)| \tag{3.4}$$

and distance measures over attributes and sets of attributes need to be defined. Too stringent a set of attributes will inhibit the ability of the approximation method to generate an approximation that is simpler than the original. At the same time, the attribute set should be sufficiently large to determine the form of B adequately, otherwise other constraints will be necessary to complete the process.

The following section proposes human perception as a basis for attribute sets. Such a basis is particularly appropriate for approximations that are to be used at larger scales, whereas at smaller scales other mathematical measures are also useful. Some methods that use purely geometrical error measures are described in Section 3.3.

3.1.5 Perceptual Properties

Studies in perception and the human visual system are helpful in constructing approximations which, being based on psychology and physiology, can provide a significant amount of information to the viewer through a small volume of data.

It is claimed that many different properties, including colour, shape and location, are registered in parallel in a pre-attentive stage of perception [**Treisman86**]. The information gained through these visual channels remains separate, and so the properties cannot be cross-related. This means, for instance, that the presence of a triangle amongst a field of circles will be noticed, as will the presence of one red object amongst many green ones, but distinguishing whether the triangle is red or green requires conscious, attentive processing (Figure 3.7 *(left)*). It is likely that the properties registered at this early stage are central to human perception.

Edge detection occurs at an even earlier stage in the visual system, and oriented edge segments play a significant part in perception, being used to build up several properties such as symmetry, convexity, area and orientation. As well as these properties, curvature, colour, line endings, contrast, brightness and closed areas have been identified as important to the identification of primitive shapes in an image [**Treisman86**].

Flow patterns and texture gradients (Figure 3.7 *(right)*) are further effects that have been shown to aid perception. Texture discrimination is described in [**Julesz81**] in terms of "textons" — elongated blobs specified by their colour, orientation, length and width (which might be identifiable through the wavelet image processing techniques described in [**Mallat89**]). This suggests that the removal of all texture-producing information in approximations may be undesirable; this fact is the basis of the pen-and-ink illustration methods mentioned in Section 1.3.

The feature-based theory of recognition attempts to describe how observers recognise objects by the identification and matching of distinctive features [**Chase86**]. Since it does not take into account the structure or organisation of the patterns in an image, it has not survived as a complete description of perception, but it does indicate how important some details may be to perception. Feature identification is an area in which automatic methods

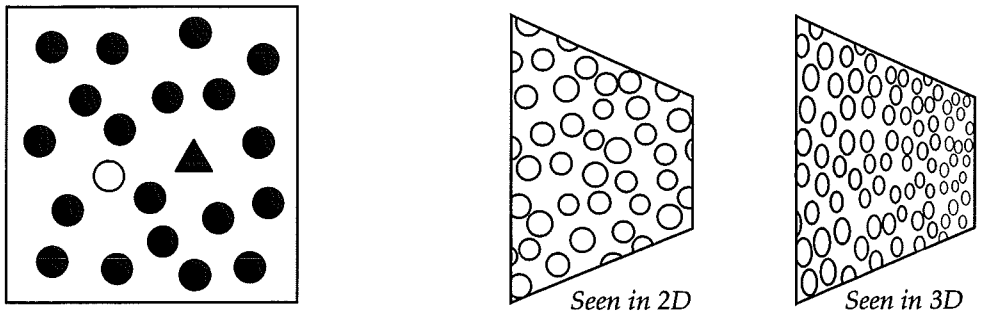


Figure 3.7: A pattern similar to those used to establish the independent pre-attentive processing of shape and colour information *(left)* and an example of how flow patterns and texture gradients influence depth perception *(right)*.

are likely to fare badly, since obtaining features which to a human characterise an object will in general be very difficult. Some of the polygon mesh simplification schemes examined in Section 3.3 attempt to preserve features such as creases and edges; nevertheless the importance of an arbitrary detail of an object remains difficult to evaluate.

3.1.6 Quality Measurements

Quality will eventually be used to assess the suitability of a particular representation of an object for generating a specific image. In this chapter, and the next, quality is associated with objects, independent of the particular image that is being generated and the rendering method in use. Chapter 5 will examine the view-dependent components of quality. As a consequence, the quality measure firstly should be based only upon properties of the object itself, but secondly should be easy to transform into image space quantities for use in a practical setting. The three most basic measures that satisfy these conditions are length, area and volume of which the last is the least satisfactory since it has no analogue in a 2D image.

When considering the quality of a particular representation another important issue is fixing a reference level against which quality can be measured. For a mathematical object such as a sphere the answer is well-defined, but for an abstract object such as a chair it is not clear what the quality of any representation is measured against. In the latter case personal judgement, or measures such as the feature size of the model, would probably be used as a basis. In practice, simply measurable quantities, such as the maximum edge length on a polygon mesh, the maximum deviation between the original and approximating surface, or even cruder measures such as the total size of the object are used. Section 5.2.2 discusses how quality variables are used to choose between different representations, while possible measurements for this task are mentioned in the discussion of the different approximation schemes in the rest of the current chapter.

3.2 Simplification

This section attempts to identify what types of objects are particularly suited to simplification and how the process might proceed. Several existing object manipulation algorithms can be re-formulated to produce simplifications, although not all of them are designed with this task in mind, and the remaining sections of this chapter look at some examples.

3.2.1 Flexibility

Not all objects are compatible with simplification. Most importantly an object must be *flexible*, since different quality representations must be possible within the same framework. Polygon meshes can, for instance, have any number of vertices and faces, or an octree can be continued recursively to different depths.

Object types may differ in the degree of simplification possible. A large polygon mesh whose vertices are being progressively removed will yield a large number of simplifications, each with only a small difference from the last. Removing items from a list of ten items will produce only a small number of probably large jumps in quality. With careful implementation, different simplifications can often be blended smoothly from one to another to minimise the "popping" that can occur with a sudden switch of representation. This is very important in order to avoid distracting users [**Yan85**] and is much easier within a simplification framework than with replacement.

3.2.2 Internal Structure

A constant level of simplification over the whole object may not be sufficient. Instead it may be useful to be able to vary the level. A landscape, for example, might be stored as a succession of grids of finer resolution. When viewed from one corner a higher resolution grid will be required in the foreground than in the distance (Figure 3.8 *(left)*). If the object is large, satisfying the requirements at one corner can have a serious impact at the opposite corner [**Floriani92**] [**Lindstrom95**]. In this case a variable density of detail, usually in the form of a *hierarchy* of detail, is essential because the size of the object is significantly larger than the size of its detail. When no hierarchical simplification is available, then splitting the object into sub-objects and simplifying each independently is preferable to proceeding with a large-scale simplification.

Simplifying an object with explicit internal structure does have a number of advantages over a similar replacement scheme that operates on a hierarchical structure. In the example of gridded landscapes, when joining together different levels of detail at boundaries it is possible to remove unwanted tears between adjacent sub-patches (Figure 3.8 *(right)*) [**Scarlatos90**].

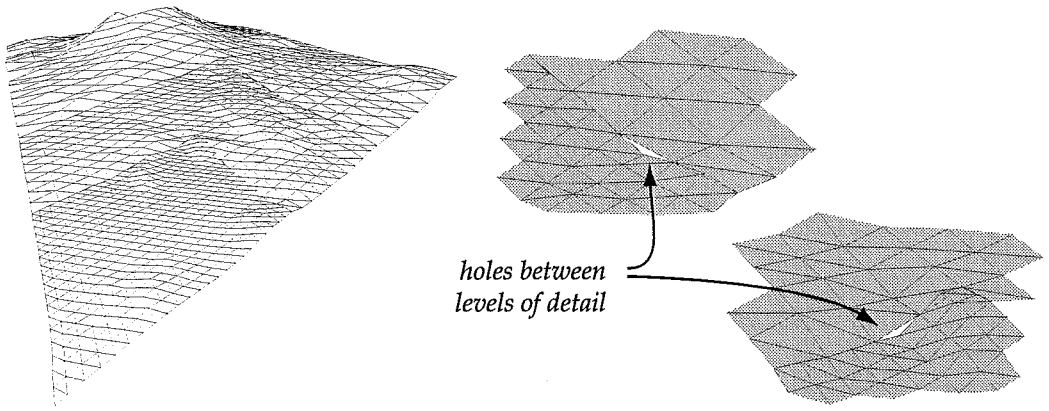


Figure 3.8: The variation in level of detail required of a medium size portion of landscape data *(left)* and a closeup of the tears that can occur if adjacent sub-patches are displayed independently *(right)*.

Hierarchical structure is another useful feature, although a note of caution is necessary since not all hierarchies are sufficiently ordered or controlled for truncation to yield good approximations. Hierarchical splines (Figure 3.22, page 64), for example, can have an arbitrarily large detail at a low level whose effect is not propagated up to higher levels — these therefore are not necessarily good simplifications. The same can be true for a list structure, which may or may not be ordered sensibly. While a good approximation will take advantage of a structured type, structure does not necessarily lead to good approximations.

Another important issue is storage space — does the simplification re-use data present at a higher level, and how much extra data is required to describe the simplification. Within simplification it is easier to share data between representations than it is in a replacement scheme.

3.2.3 Simplification and Quality Measurement

The rigid structure of simplification lends itself to measuring changes in quality. This is usually due to being able to compare two objects of the same type with each other, providing a natural mapping over which to measure differences. For instance, when simplifying a polygon mesh, different error metrics can be defined in terms of the distance between the original surface and the simplification, and can almost always be performed locally. A good metric might be the maximum over the original surface of the distance from a point on the original to the closest point on the simplification.

3.3 Polygon Mesh Models

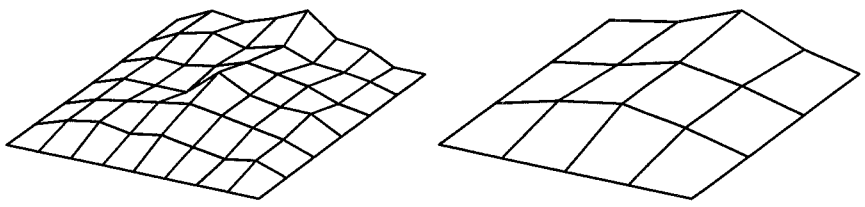
Much attention has been paid to polygon meshes, not only because they are a popular tool for flexible modelling and fast rendering, but also because the results of automatic data capture systems, such as 3D scanners, can often be converted immediately into a regular mesh structure.

Polygon meshes, or more commonly triangle meshes, often occur not because the user wishes to model an object with a surface made up from polygonal faces, but because they are a convenient approximation to a curved surface. Algorithms have been required to remove the vast numbers of unnecessary vertices whilst maintaining the accuracy of surface. More recently, these algorithms have been extended to generate coarse levels of details for polygon meshes explicitly, where accuracy is no longer kept as high. This section includes short descriptions of seven existing algorithms, and a discussion of their relative merits in the context of large-scale simplification.

Because arbitrary surfaces can be represented, including holes, loops and separated mesh sections, it needs to be decided whether each method should, or should not, preserve the topology of the input mesh. It can be argued either that details such as holes and handles are important to perceiving the object, or that they are indeed just minor details and should be removed. While the latter is more in line with the attitude to simplification taken in this dissertation, the former might be important in some contexts, for example when an engineer needs to check the presence of bolt holes in an aircraft part.

3.3.1 Sub-sampling Regular Grids

If the polygon mesh is based on a regular topology, such as a rectangular grid, an obvious approach is to sub-sample every n^{th} grid point in each dimension, reducing the size by a factor of n^2 (Figure 3.9). Sub-Sampling is essentially a filtering process and better results can be achieved if the samples are based on weighted sums of the surrounding vertices rather than a single sample point, since aliasing effects are otherwise noticeable **[Williams83]**.



Original regular grid Subsampled by factor of 2

Figure 3.9: A rectangular mesh reduced by sub-sampling every other vertex.

This method inherits the advantages of the regular grid structure and is very space efficient

if no filtering is used, since the different levels can be generated on the fly. The restriction to regular grids can be a big disadvantage for large grids, since it forces all areas to be sampled at the same density, so adaptive methods are often used (*e.g.* [**DeHaemer91**] [**Falby93**]).

3.3.2 Decimation of Triangle Meshes

The most straightforward approach for *general* triangle meshes is described in [**Schroeder92**]. The same stage is repeated until the desired results are achieved — a surface element, in this case a vertex, is identified and removed, after which a local restructuring step creates a simpler mesh. The restructuring involves re-triangulating the hole where the vertex has been removed (Figure 3.10) with constraints to generate a well-shaped triangulation and preserve any crease edges and topology. Crease edges are identified in advance by examining the surrounding polygons. A **decimation criterion** is computed for each vertex based on its distance from the average plane formed by the neighbouring vertices, unless the vertex is on an edge when its distance from the average edge line is calculated instead. The criterion is used to determine whether a vertex should be removed or not, and ensures that important vertices remain in the mesh.

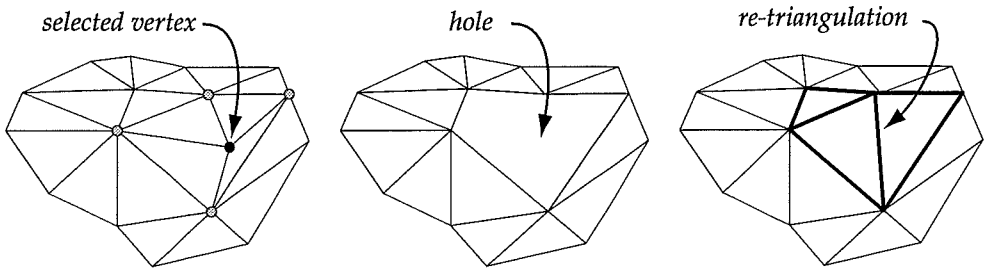


Figure 3.10: Illustration of the removal of a single vertex during the decimation process.

Suggested error measures include: the distance of removed vertices from their replacement; the change in surface area or volume from the original mesh to the replacement, both of which can be measured locally; and an energy function that depends on the location of the vertices.

3.3.3 Polygon Grouping

The method described in [**Hinker93**] is based on the identification of nearly co-planar sets of triangles which are simplified along their boundaries and then have their interiors re-triangulated. The sets are identified by grouping triangles whose normals differ only by

an angle less than a specified amount; several ways of performing this are described. It is likely that this method would not perform well for large-scale simplification due to the difficulty in identifying what constitutes a near-planar set.

3.3.4 Re-Sampling Polygonal Surfaces

A method designed explicitly for generating levels of detail is given in [**Turk92**]. A re-sampling approach creates a new mesh with the required number of vertices all lying on the original surface and possessing the same topological features. Sample points are distributed over the surface randomly, and then point repulsion is used to achieve a good sampling, weighted so that more points are placed in areas of higher curvature. The new sample points are then merged into the existing mesh after which the original vertices are removed (Figure 3.11) using a method similar to Section 3.3.2.

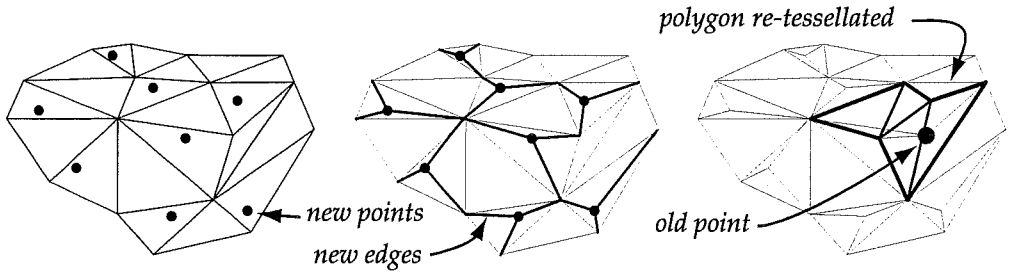


Figure 3.11: Illustration of polygon mesh re-sampling process showing the introduction of new vertices and the removal of old ones.

The method can be adapted to generate multiple levels of detail sharing vertices between levels by progressing from coarse to fine densities, forcing the vertices from the previous levels to be included in the next one. Blending between one level and the next can be achieved by moving the new vertices out from the previous surface until they reach the desired positions.

3.3.5 Mesh Optimisation

The work of [**Hoppe93**] has been driven by the need to create high-fidelity meshes with a minimum number of triangles for datasets obtained from 3D scanning devices. The constraints on the process are formalised into an energy function, which includes a term representing the error between the surface and the original sample points. Basic local mesh operations of edge collapse, edge swap and edge split (Figure 3.12), all of which maintain the topological form of the mesh, are used to modify the mesh and minimise this energy.

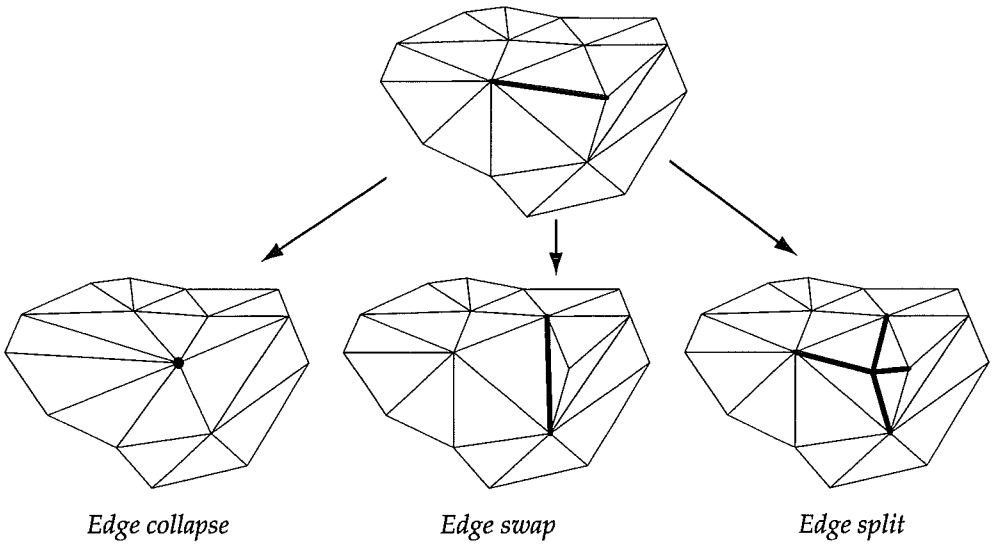


Figure 3.12: The basic operations used to alter the mesh to minimise the energy function.

The optimisation is performed by repeating two stages. The first uses a random descent method to minimise energy while performing mesh operations to alter the mesh structure but keeping the vertices fixed. By examining their projection onto the original surface, the second stage moves the vertices so that the energy of the new surface is minimised. Although expensive, the results are very impressive, and capture high curvature regions effectively while not oversampling those of low curvature.

3.3.6 Meshed Object Reduction

The method presented in [**Rossignac92**] is intended to create models rapidly that will be fast to render and that, while maintaining some of the features of the originals, do *not* necessarily preserve their topology, and is in use in the IBM 3D Interaction Accelerator [**Borrel95**]. The first stage of the process forms clusters of vertices based on spatial locality by superimposing a uniform grid on the object and grouping all vertices that fall in each cell. The second stage examines the effect this would have on the faces in the mesh, removing or simplifying them accordingly (Figure 3.13). Using this procedure, triangles might be collapsed into either points or lines if two or more vertices fall in the same cluster. Since multiple faces may collapse onto the same polygon, redundant lines, points or triangles have to be identified and removed.

Blending between levels of detail can be achieved by interpolating between the position of a vertex at one level and the clustered vertex location at the next. The method is not restricted to connected triangle meshes, but could also be used for disconnected sets of faces. The reg-

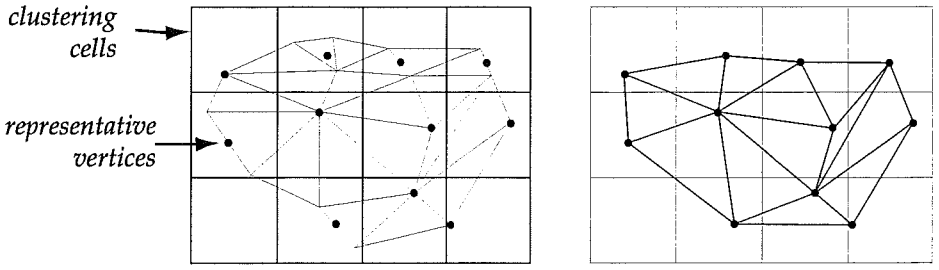


Figure 3.13: The clustering algorithm applied in two dimensions.

ular grid induces aliasing problems — the approximations are very unstable when it comes to translating or rotating the original object before simplification, and features smaller than the grid spacing are likely to collapse to points or lines. There is also no attempt to take into account areas of, say, higher curvature, since this clustering is independent of the actual object.

3.3.7 Wavelet Decomposition of Arbitrary Surfaces

Rather than using the existing mesh geometry as a starting point for simplification, wavelet decomposition [**Lounsberry94**] requires the mesh to be described in a hierarchical manner. Wavelet analysis provides a mathematical background which can be used to show that the representations derived have the minimum error according to a given error metric. While developing the use of wavelets for surfaces in a general setting, the implementation described is based on polyhedral subdivision surfaces using the triangular scheme shown in Figure 3.14 *(left)*.

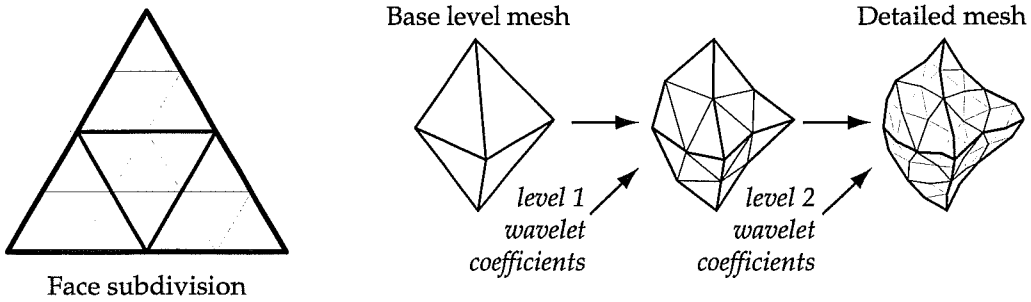


Figure 3.14: The triangular subdivision scheme which forms the basis of the wavelet mesh description *(left)* and the construction of a wavelet decomposition of a mesh *(right)*.

A base mesh is stored together with a set of coefficients that describe modifications to this

mesh (Figure 3.14 *(right)*) which can be ordered by the contribution they make to the surface, so picking the first n coefficients should always yield a best approximation. Furthermore, these changes can be blended in smoothly by scaling the coefficients from zero to their full value.

The analysis proceeds on a subdivision surface formed over each face of the base mesh. This can be obtained from an arbitrary mesh using the method given in **[Eck95]**. Face sites are inserted dynamically onto the surface and expanded until the surface is covered by regions each containing a single face site and meeting conditions that make these suitable to be faces of the base mesh. These are then processed to form the base mesh (Figure 3.15) on which the subdivision surface can be built by re-sampling the original surface at the required points. This method, although costly, could also be used for other situations that require a subdivision mesh as input.

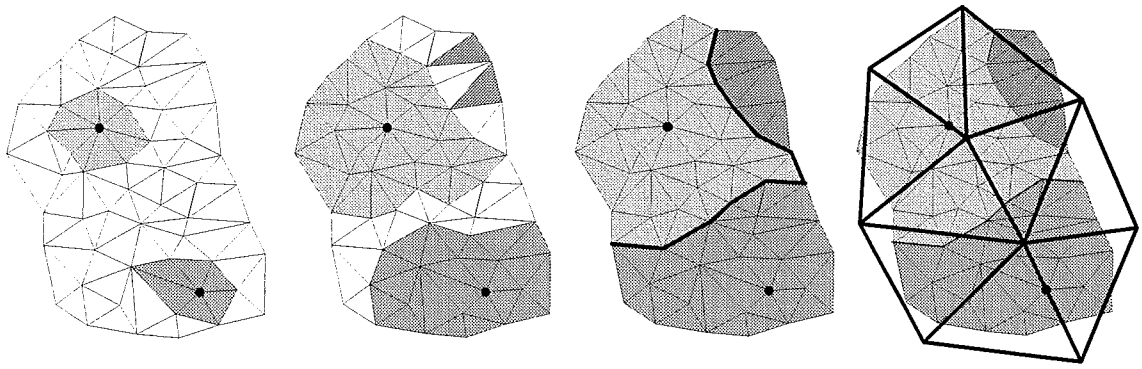


Figure 3.15: Outline of the construction of a subdivision base mesh from an arbitrary mesh.

3.3.8 Discussion

Some of the important features of these methods are summarised in the table below.

	input	*preserves topology*	*quality control*	*error measure*	*effect*	*memory*	*complexity*
Sub-sampling	subdivision surface	fixed	few levels	×	smooths	very low	very low
Decimation	triangle mesh	✓	error threshold	vertex-surface distance	preserves edges	medium	medium
Grouping	triangle mesh	✓	error threshold	normal variation	difficult to simplify greatly	high	medium
Re-sampling	triangle mesh	✓	number of points	×	smooths	low	medium
Optimisation	triangle mesh	✓	error threshold	vertex-surface distance	good	high	very high
Reduction	triangle mesh	×	few levels	×	generates lines and points	low	low
Wavelets	subdivision surface	✓	error threshold	surface-surface distance	good	medium	high

The potential disadvantage of many of the higher quality algorithms is their topology preserving nature, which means, for example, that when simplifying an object that contains many holes or a large number of pronounced geometrical features (Figure 3.16) there is a limit to the level of simplification that can be achieved.

Few methods are designed to work well when a large degree of reduction is required, since they are mainly aimed at removing redundant vertices rather than achieving rough approximation. Several techniques, particularly sub-sampling methods, tend to smooth out details, although the decimation scheme attempts to identify and preserve edges and creases in the mesh. While mesh reduction aims to remove such details, this method has the disadvantage that the models produced are no longer strictly polygon meshes, due to the inclusion of point and line features. While these work well as approximations at small sizes, the effect of the line at larger scales of approximation is undesirable, as can be seen in Figure 3.17. The identification of such "line-like" features in the object could be used to gener-

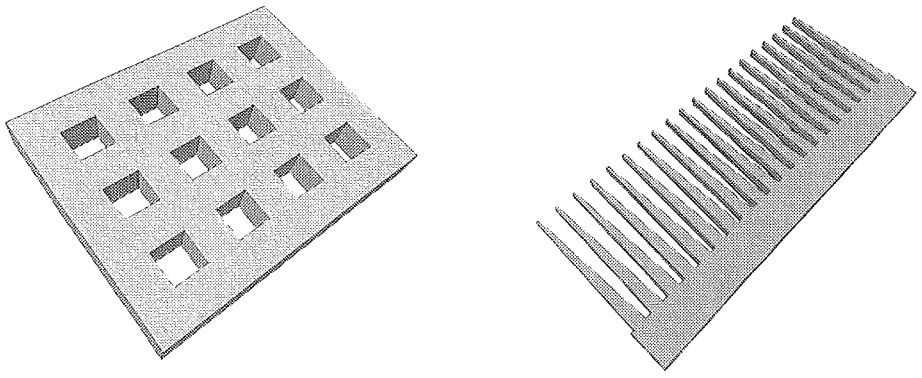


Figure 3.16: A drilled plate that cannot be greatly simplified if the original topology is maintained *(left)* and a comb-like structure with similar problems *(right)*.

ate better simplifications (or more likely replacements). Lines and points also cause problems for some rendering methods, such as ray-tracing, that rely on point sampling.

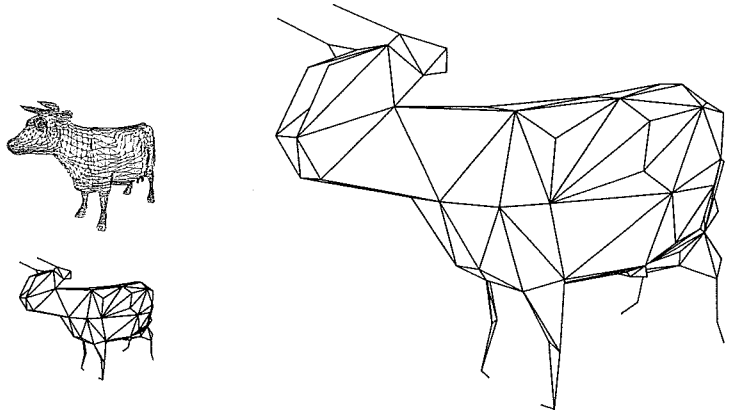


Figure 3.17: Large scale reduction introduces lines as primitives which look satisfactory at small scales *(left)*, but fail to capture the original object at larger scales *(right)*.

As far as speed is concerned the reduction approach is very fast if the mesh is stored suitably, and is designed to work in real time. The impressive results obtained by the wavelet analysis, and mesh optimisation in particular, require a huge amount of computation. Several of the methods allow fine control over the number and quality of the levels of detail provided but the decimation scheme, and more particularly the grouping method are limited in the maximum simplification achievable. All methods allow interpolating between different levels of detail with minor amendments.

All these methods are designed to solve the general problems of polygon mesh simplifica-

tion, and can be usefully employed to generate simplifications for use in an approximation system. Much more successful schemes are possible in restricted cases, such as terrain modelling where the problem is reduced to a two-dimensional one (*e.g.* **[Brown95]** **[Floriani92]**). The next section looks at some other situations in which knowledge of the source of the data is advantageous.

3.4 Parametric Models

While polygon mesh simplification techniques can be effective, they are hampered by lacking higher-level knowledge about the object being simplified. In many cases, polygon meshes are the results of some higher-level modelling process, and applying simplification to this process *then* polygonalising (Figure 3.18) is simpler and produces better approximations.

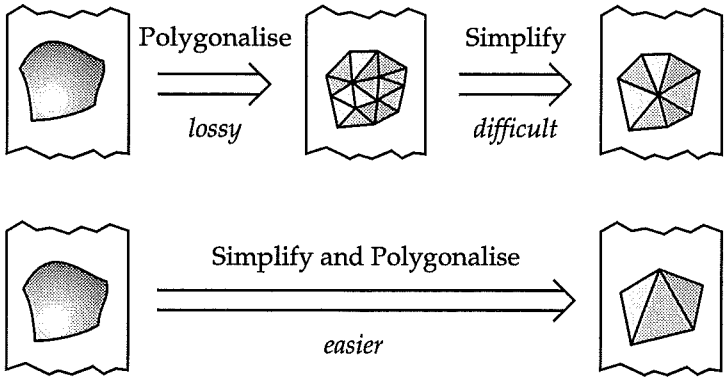


Figure 3.18: The ordering of simplification and polygonalisation can affect the quality and speed of simplification.

When an object is described at a higher-level than it will be used, an **expansion** operation has to occur. Since the expanded representation is generated in a known manner, it should be possible to use the additional information present to create good simplifications with much less effort. Generative modelling tools and curved surface modellers are two examples where the form of the output can be controlled directly.

3.4.1 Generative Modelling

Many higher level modelling methods can be described in terms of **generative operators** **[Snyder92]**. These act on simpler primitives, which might themselves be the results of generative operations, to produce an object built up recursively from a tree of operators with parameters and data sources at the leaves (Figure 3.19).

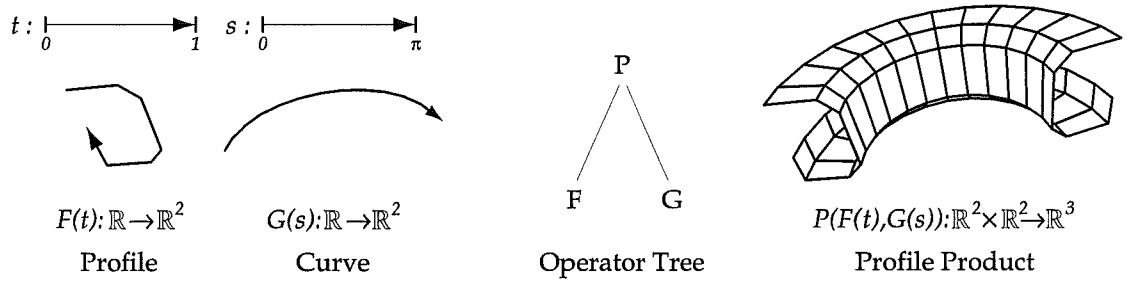


Figure 3.19: A model *(right)* constructed by the application of a generative operator to two simpler shapes *(left)*.

Two aspects of this process can be simplified whilst keeping the structure of the operator composition the same. The first is the data sources at the leaves of the hierarchy; the second is the action of the operators. A simple example is the polygonalisation of a cylinder which can be viewed as either the extrusion of a circle along an axis, or the rotation of a profile around an axis. In these cases, the profile or the cross-section form the simpler primitive, while the generative operations are extrusion or rotation (Figure 3.20).

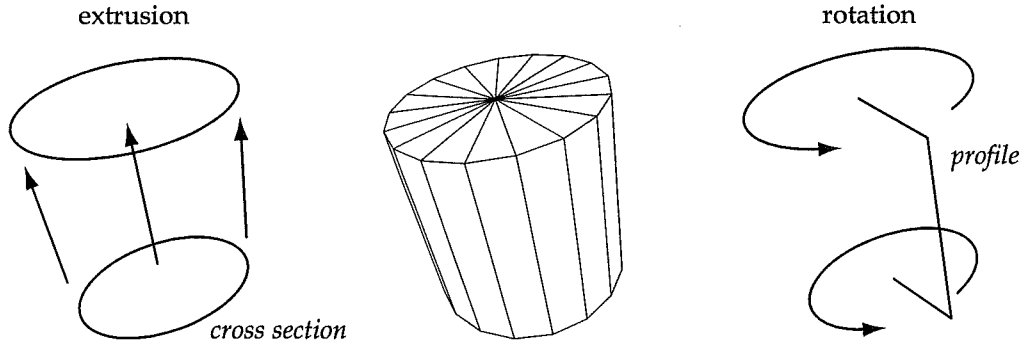


Figure 3.20: A cylinder defined in two ways by generative operations.

As can be seen in the example, the operators, while defined continuously, have to have their output **sampled** in order to generate a polygonal representation. Similarly, the circle forming the cross-section or the profile curve has to be sampled in order to convert it to a line-chain approximation.

3.4.2 Curved Surface Modelling

Parametric curved surface descriptions (*e.g.* **[Foley90]**) are another flexible surface specification method. While a few rendering algorithms can cope directly with such primitives, most use a polygonalisation at some stage. The size and quality of this polygonalisation can be varied to produce multiple levels of detail. Surfaces such as **tensor product splines** are

described by a mapping from **parameter space** to the location of the surface in 3D. Sampling is usually performed in parameter space, either uniformly, which may result in an uneven sampling on the actual surface, or adaptively, in which case the curvature of the surface is used to control the sampling frequency in the parameter plane (Figure 3.21).

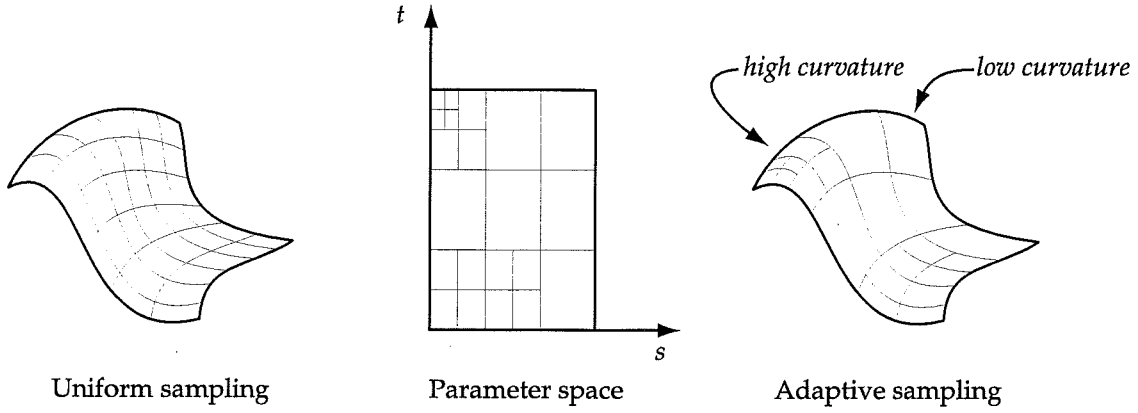


Figure 3.21: The relationship between sampling in the parameter space *(centre)* and the resulting sampling of the surface for uniform *(left)* and adaptive sampling *(right)*.

Hierarchical splines [Forsey91] are a more recent description which provide a multi-resolution description of the surface itself, independent of the polygonalisation. These solve some problems with tensor product spline surfaces which are affected globally whenever there is a local addition of detail. A hierarchical B-spline surface is built up from a number of layers of increasing detail which locally modify the coarser surface. If this addition is done in the coordinate frame of the local surface, then editing at different scales becomes more intuitive. The formulation ensures that all the detail elements merge smoothly into the surface around their edges.

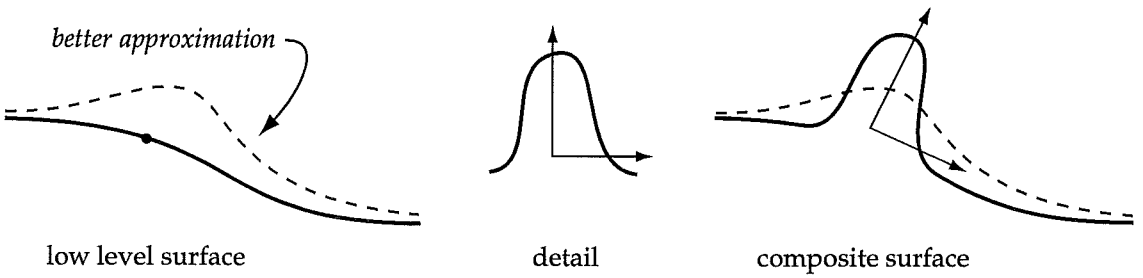
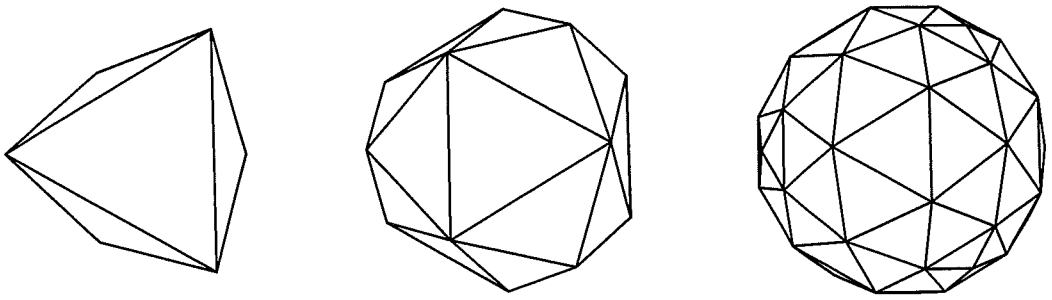


Figure 3.22: A hierarchical spline curve in two dimensions showing a low level surface *(left)* and a detail modification *(centre)* and the composite surface *(right)* which is not particularly well approximated by the low level one.

While hierarchical splines provide a neat tool for storing and editing surfaces at different scales, they have a drawback when it comes to removing details for approximation — there are no constraints in the system that force the truncated representation to be a good approximation to the full one (Figure 3.22).

3.4.3 Subdivision Schemes

In contrast to the earlier example of a cylinder, a sphere is usually converted to polygons as a subdivision surface, which produces more evenly sized triangles than a simple generative approach. The subdivision surface is based on either a cube or an octahedron with all new points introduced being re-projected onto the surface of the sphere by normalisation. Such a representation can also be generated at arbitrary levels of detail (Figure 3.23).



Base level One level of subdivision Two levels of subdivision

Figure 3.23: A sphere as the limit of a subdivision surface based on an octahedron.

Although there are no *explicit* parameters, the structural subdivision enables the surface to be described at a range of qualities, and can be viewed as an alternative sampling in the parameter space of each polygon in the base mesh. As an aside, it is very rarely taken into account that such a description of a sphere always underestimates both volume and surface area, and that certainly at low levels of subdivision a small scaling correction should be applied if the size of the sphere is to be maintained.

The **Wavelet surfaces [Lounsberry94]** described in Section 3.3.7 are based on subdivision schemes and provide a hierarchical description of surfaces. In contrast to hierarchical splines, the wavelet scheme includes the constraint that the higher-level coarser surfaces are not independent of details, but change to accommodate them to ensure a good approximation, a fact also exploited in **[Vemuri94]** for generating hierarchically deformable superquadric objects.

3.4.4 Simplifying Parametric Models

Both generative models and curved surfaces are examples of parametric generation methods. Subdivision surfaces also hide a parametrisation over the base mesh, and all three can be described in terms of a continuous mapping,

$$M(t) : \mathbb{P} \rightarrow \mathbb{R}^3 \tag{3.5}$$

from parameter space to object space. Creating a polygonal representation involves a **sampling** of parameter space $\mathbb{P}$. The simplest approach is to sample $\mathbb{P}$ **uniformly**. Different sample spacings can be used to generate multiple results of different qualities. A uniform sampling in parameter space will not however guarantee a uniform *quality* result in object space. This can be seen in the case of the parametric surface patch whose varying curvature can cause a significant variation in quality (Figure 3.21, page 64). The choice of sample spacing is often guessed without analysing the effect in object space. Figure 3.24 shows an example of simplification performed by altering the sampling rate on two parametric models.

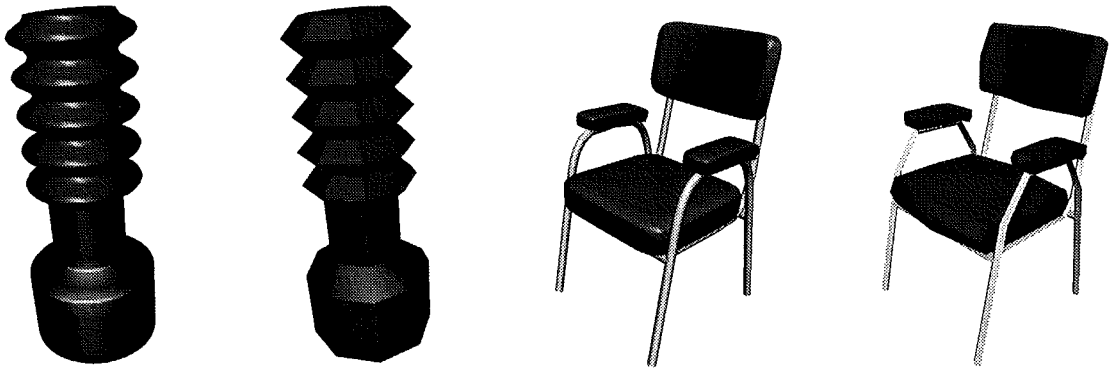


Figure 3.24: Simplifications of two objects defined in terms of generative operators, reducing the number of triangles to 8.6 % *(left)* and 5.8 % *(right)* of the totals.

By sampling **adaptively** in parameter space, more control is available over the local quality of the results. Unfortunately, computing the level of sampling required over a surface is not trivial. Some representations possess properties that enable error bounds to be computed without sampling. Bézier surfaces, for example, always lie in the convex hull of their control points and so are constrained by the physical dimensions of this volume [**Foley90**]. The **interval arithmetic** analysis that can be performed on generative models [**Duff92**] [**Snyder92**] is another example where maximum errors can be predicted. By evaluating errors on input data, and the effect of operators on that data, the errors of the output can be bounded. A further disadvantage with adaptive sampling is that the non-uniform sampling causes an irregular network of polygons which has to be dealt with carefully to avoid T-vertices where different sampling densities meet.

An even quality over the surface still does not guarantee a high-quality approximation. For instance, the typical polyhedral approximation to a sphere (Figure 3.23 *(left)*) always underestimates the total size, even though the surface is within a certain distance of the sphere — the obvious mathematical error measures are not always ideal. Compared to the polygon mesh simplification methods, simplifying a parametric model is much easier and the results are of higher quality per polygon in the final result.

3.5 Volume-Based Models

The previous sections have concentrated on surface representations of objects. This section examines some volumetric representations that are amenable to simplification. Volume-based data is often simpler to deal with using standard mathematical approximation techniques because such data is based on a simple function, or set of functions,

$$f(\boldsymbol{x}) \qquad \text{for } x \in D \subset \mathbb{R}^3 \tag{3.6}$$

defined directly over a finite region of 3D space. Surface data, on the other hand, typically requires a parametric mapping into object space, and this mapping causes many problems for analysis.

By expressing $f(\boldsymbol{x})$ in terms of a finite set of basis functions $B_i(\boldsymbol{x})$ as

$$f(\boldsymbol{x}) = \sum_i f_i \cdot B_i(\boldsymbol{x}) \tag{3.7}$$

the function can be approximated by a finite set of sample values f_i. Most commonly the function f represents density, and can either be a continuous function in the range $[0, 1]$ or a discrete function taking the value 0 or 1. A discrete density is often constructed from a continuous one by **thresholding**,

$$f_T(\boldsymbol{x}) = \begin{cases} 0 & \text{if } f(\boldsymbol{x}) < T \\ 1 & \text{otherwise} \end{cases} \tag{3.8}$$

3.5.1 Uniform Voxel and Octree Storage

As an example, the basis functions to describe density over **uniform voxels** (*e.g.* **[Foley90]**), are the set of box functions (Figure 3.25),

$$B_i(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ is in } C_i, \text{ the unit cube centred at } \boldsymbol{c}_i \\ 0 & \text{elsewhere} \end{cases} \tag{3.9}$$

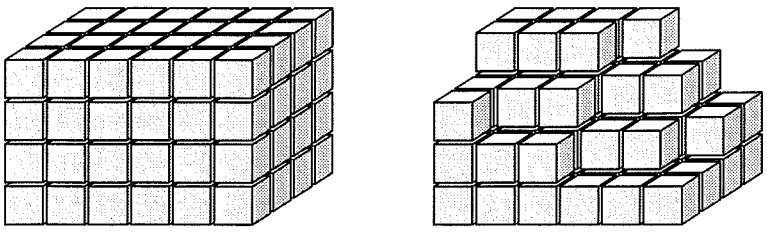
where $\boldsymbol{c}_i$ is a set of points on a unit lattice. The value of $f(\boldsymbol{x})$ can be reconstructed efficiently because these basis functions do not overlap so that each point in space is defined by at most

one sample value, f_i. This also means that the sample values can be found as

$$f_i = < B_i, f > \tag{3.10}$$

$$= \int B_i(\boldsymbol{x}) f(\boldsymbol{x})\, d\boldsymbol{x} \Big/ \int B_i(\boldsymbol{x})\, d\boldsymbol{x} \tag{3.11}$$

$$= \int_{C_i} f(\boldsymbol{x})\, d\boldsymbol{x} \Big/ \int_{C_i} d\boldsymbol{x} \tag{3.12}$$



UniformVoxel Grid Example Object

Figure 3.25: The basis functions for volume data over a uniform voxel domain and an example object formed by combining some of them.

Basis functions can be defined for non-uniform voxels and also for octrees. A **uniform octree** [**Samet90**], can be represented as a 8-way tree of values which each correspond to a basis function (Figure 3.26). The basis function at the root is a unit cube, and the basis functions of the children of a node are cubes half the width of the parent, positioned to cover the parent without overlapping. At level l the cubes are therefore 2^{-l} times as wide as the root cube.

Only the leaf nodes are strictly necessary to describe the original function, but the inclusion of internal nodes allows the hierarchy to be terminated at higher levels, leaving the new leaf nodes describing the average value within their domain. This immediately gives rise to a simplification of the original object, consisting of *any* subtree (Figure 3.27); it also means that simplification does not have to be applied uniformly over the object, since different branches may be truncated at different depths.

Multi-dimensional trees [**Wilhelms94**] extend the notion of irregular octrees to include more general approximations at the internal nodes. An error metric allows quality to be determined for any truncated hierarchy and enables the efficient calculation of good approximations.

3.5.2 Wavelet Representations

Wavelet transforms can also be used for storing volumetric data [**Muraki93**]. As in the case of the octree, discarding some of the coefficients from the representation gives a lower quality encoding. The octree encoding is very close to the Haar wavelet basis [**Fournier94**],

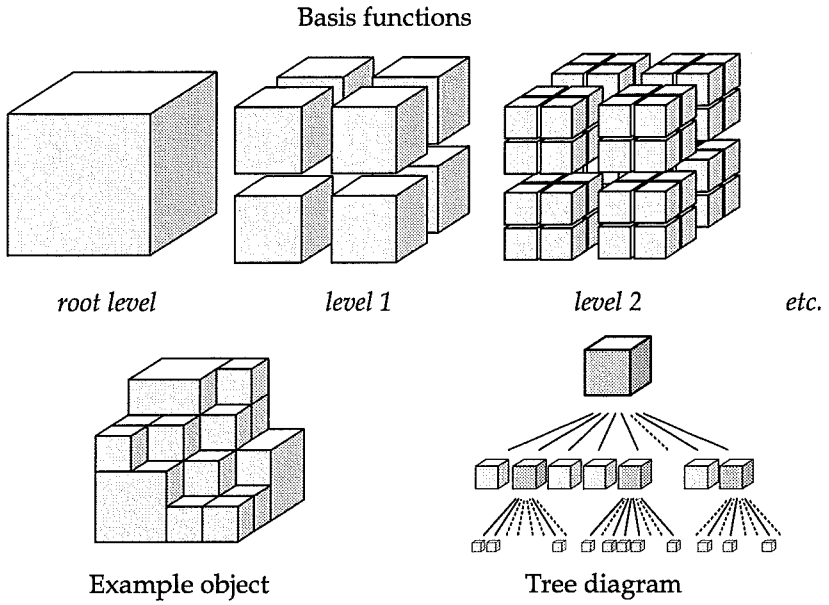


Figure 3.26: Basis functions for an octree representation and a tree diagram of an example structure.

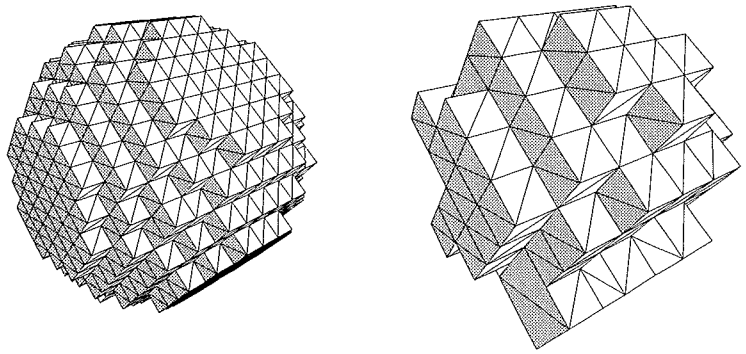


Figure 3.27: An octree representation of an object truncated at two different depths.

but by using more complex wavelets, much better results can be obtained so that the density function can be represented to the same accuracy with many fewer wavelet coefficients. Moreover, the errors are much less noticeable, since by using smooth overlapping basis functions, the reconstruction does not exhibit the sharp rectangular edges visible in Figure 3.27. This scheme is particularly amenable to the transmission of large volumetric datasets over slow networks, since the object can be viewed before the entire data set is fetched. Unfortunately, it has higher computational demands than the multi-dimensional trees approach.

3.5.3 Potential Fields

Another representation bearing some similarity to wavelet volumes, but which has been developed primarily to describe discrete rather than continuous densities, is the "blobby model" [**Muraki91**]. A blobby model is an implicit surface, defined by an iso-value of a **potential field**, which like density data, is a scalar valued function

$$p(\boldsymbol{x}) \qquad \text{for } x \in D \subset \mathbb{R}^3 \,. \tag{3.13}$$

Thresholding this function defines an object which is solid where $p(\boldsymbol{x}) \geq T$, and, assuming p is continuous, has a surface where $p(\boldsymbol{x}) = T$.

The potential function is constructed as a sum of primitive functions translated to sample points $\boldsymbol{p}_i$ and scaled by weights w_i

$$p(\boldsymbol{x}) = \sum w_i \, q(\boldsymbol{x} - \boldsymbol{p}_i) \tag{3.14}$$

The weights therefore describe the contribution that each sample makes to the total potential before thresholding. For a simple exponentially decaying primitive function $q(\boldsymbol{x})$, such as

$$q(\boldsymbol{x}) = e^{-a|\boldsymbol{x}|^2} \tag{3.15}$$

the weighting simply increases the radius of the field, and is a measure of the local size of the resulting surface. Sorting these coefficients and then discarding the smaller ones will therefore yield a straightforward simplification of the object.

3.5.4 Discussion

The previous methods are just a few that can be applied to volumetric data. Due to the simple representation, many other mathematical schemes, such as Fourier transformations, might also be used to represent such functions. If a suitable ordering can be placed on the discrete samples or coefficients that constitute this type of representation, then it is possible to generate simplifications of the objects. In order to have control over the quality of these approximations, it is necessary to be able to perform a certain amount of analysis on the representation in use. With volumetric data, this task is often made simpler by the straightforward mapping between the representations and the space in which the objects reside, allowing direct comparisons of values at all points in space.

One of the main difficulties with volume data is how to render it. Current methods tend to favour one of two approaches, either a ray-tracing based solution (*e.g.* [**Upson88**] [**Blinn82**]), sometimes involving numerical techniques, or the conversion of the data to some other representation such as polygonal models [**Lorensen87**], texture-mapped primitives [**Wilson94**] or through Bézier clipping [**Nishita94**]. These are a case of approximation by replacement, the subject of the following chapter. Recently, however, there has also been interest in approximating complex objects defined using boundary representations and CSG by simple volumetric objects for rapid previewing [**Shareef95**].

3.6 Summary

This chapter has introduced simplification and replacement as two classes of object approximation methods. It has emphasised that it is important to generate approximations that can be used at large scales as well as small, and has suggested that techniques based on perceptual attributes are a good basis for such methods. Existing algorithms for polygon mesh simplification have been examined in the light of the large scale reduction necessary to generate hierarchical approximations, and the advantages of simplifying models at earlier stages, perhaps as a parametric primitive, have been described.

Like parametric models, algorithmic and procedural modelling paradigms (*e.g.* **[Amburn86]** **[Tommasi90]** **[Reeves83]**) offer plenty of scope for automatically generating approximations. Since such methods of rapidly introducing detail will become essential as more complex models are required, an important area of research will be concerned with the automatic creation of approximations as, or before, the objects themselves are generated.

Chapter 4

Object Replacement

— In which new methods for generating replacements of groups of objects are introduced.

This chapter continues the examination of approximation by proposing some new methods for replacing an object, or a group of objects, by another. The first section presents an overview of the replacement process and looks at how to use attribute matching (Section 3.1.4) to calculate geometric replacements. Sections 4.2 and 4.3 describe original techniques for calculating attribute values and generating replacement objects which match these attributes. These sections also compare the results with some other replacement schemes. Section 4.4 studies the approximation of surface properties, and illumination models. In addition, it includes descriptions of some more unusual replacements, such as "image-map" objects. The final section proposes ways in which automatic structuring algorithms can be modified to choose which sets of objects to group together for replacement.

4.1 Object Replacement

A replacement algorithm has to decide which *type* of object would make a good replacement, and establish the form of the chosen approximation. In order to do this it is necessary to be able to make a comparison between objects of different types. The replacement schemes described in this chapter rely greatly on **attribute matching**.

As with simplification, theories of perception are a good source of information, since approximation schemes that attempt to emulate the human process of object recognition are

likely to be more effective than those that do not. Two results of particular interest are described in this section. The first concerns a theory of recognition based upon the hierarchical decomposition of objects and their identification in terms of simplified forms. The second describes a set of primitive attributes thought to play an important part in the recognition of individual forms.

4.1.1 Perception Based on Decomposition

Recognition by components (RBC) describes object perception in terms of decomposing objects into geometric primitives or **geons** [**Biederman85**]. Thirty-six geons are generated by changing the qualitative parameters describing a **generalised cone** (Figure 4.1). The parameters are related to the visual system's ability to detect properties such as symmetry or curvature in the edges of primitives, even in a perspective image.

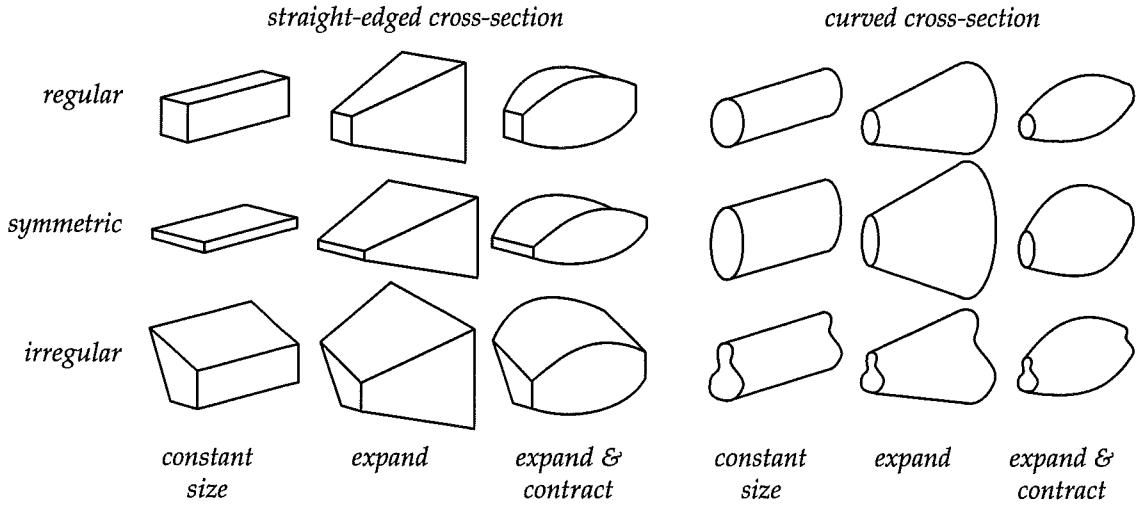


Figure 4.1: Eighteen of the geons used in RBC and the properties by which they are described and recognised. The other eighteen are formed from these cross-sections swept along a curved axis.

RBC postulates that the visual system rapidly segments a complex object into geons by identifying joins in its image. These typically occur at sharp concavities or discontinuities in the silhouette of the object. The parts obtained are matched rapidly, via the parameters describing them, to primitive geons, and the strongest matching primitive is chosen — a process mirrored by hierarchical approximation.

The presence of a property, such as curvature or symmetry, in the 2D image indicates the same property in the 3D object. Additional properties, such as relative size along axes, position, and orientation, are obtained and used to aid object identification. Difficulties can occur in recognising objects from unusual angles, where the rapid matching of primitives

becomes difficult. RBC proposes that an internal model of an object is stored in a "canonical orientation" which maximises the ability to identify the object components.

One drawback is that this theory is only applicable to so-called discrete objects such as bricks or chairs, as opposed to amorphous objects such as sand or water — many amorphous, natural objects would seem to pose difficulties. Biederman proposes that these are recognised via texture or colour alone, two features which are not required for the basic RBC mechanism which asserts that the boundary properties are much more useful and important than colour or texture — humans can identify a chair, for example, whatever colour it is painted.

The results of RBC imply that preserving a suitable set of properties during object replacement will lead to a high degree of recognisability. The challenge is to identify such properties and find a means of calculating them.

4.1.2 Visual Attribute Matching

Perceptual theories are based upon *image-space* properties, which are derived from corresponding *object-space* attributes — the brain learns to make this association between image and object-space. Since approximation can occur independently from viewing, attributes for replacement need to be expressed in terms of object-space quantities.

The primary visual attributes considered in this chapter are **location**, **size**, **shape**, and **colour**. These are chosen for two reasons :- for their perceptual importance, and also from consideration of the calculation of attribute values and replacement objects. Location, size and average colour are very important cues in distinguishing objects — a gross discrepancy between objects and replacements in these attributes is immediately noticeable, even when replacement occurs at small scales.

The size of an opaque object in an image is related to its visible surface area. Colour is derived from the surface properties of the object together with the surface geometry. Lighting also affects colour, but is independent of the object and therefore is not used in this context. At small scales, shape is not important, since factors such as pixel size swamp any indication of shape. As replacement occurs at larger scales, hints about shape are a great aid to recognition. Shape is very difficult to quantify and encode simply, and the measures used in this chapter choose two of the most significant features, **orientation** and **aspect ratio**, to represent shape [**Rock86**]. Other features that might be considered include axes of symmetry and the location of corners, curvature and convexity in the silhouette of the object [**Biederman85**].

4.2 Computing Attribute Values

Even though attribute values will be computed in object space, it must not be forgotten that the approximation will be judged by image-space properties. For instance, a crate of books will look the same whether or not it actually contains the books. Visibility is therefore a very important aspect of an object-space analysis. This section introduces two separate approaches to calculating attribute values — **surface traversal** and **object sampling**. Both of these rely on accumulating attribute values and then averaging them. It should be noted that exact answers are not necessarily required in this context, and that, when calculating attribute values, full use should be made of existing approximations, rather than objects with maximum detail.

4.2.1 Visibility

Much of the analysis performed in this section is concerned with the *surface* of an object, rather than its *volume* because, for an opaque object, only the surface is visible (from viewpoints external to the object). The same is true, to some extent, for semi-transparent objects since internal points always lie underneath a surface point when projected onto an image plane. Thus the *area* occupied by an object on the image plane is dependent only on its surface (Figure 4.2).

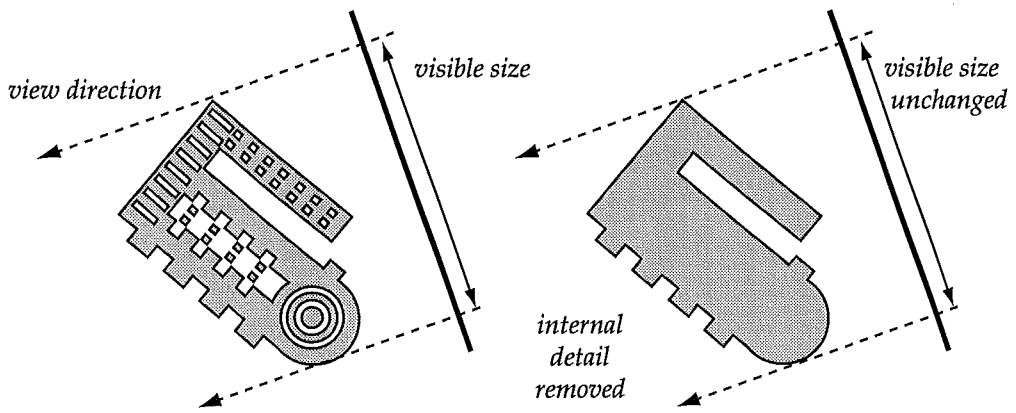


Figure 4.2: A 2D illustration that the visible size of an object depends on its surface rather than its interior.

Although volume-based measurements may be more straightforward, they do not reflect the values being computed. In particular they place little emphasis on thin regions with little volume but a large surface area (such as a single polygon.)

Unfortunately not all surface regions necessarily contribute equally to the image. This is due to self-occlusion between different regions of the object, and is relatively difficult to detect and measure. For each point x on a surface S of a completely opaque object $\mathcal{V}$, a

visibility function, $V_S(x, \hat{n})$ can be defined which indicates the visibility of that point from direction $\hat{n}$ (Figure 4.3 *(left)*),

$$V_S(x, \hat{n}) := \begin{cases} 1 & \text{if point } x \text{ is visible from direction } \hat{n}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

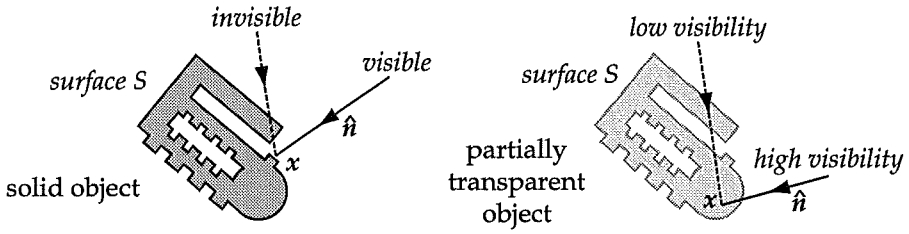


Figure 4.3: The visibility function for a point x on a surface S as a function of the viewing direction $\hat{n}$ for opaque *(left)* and semi-transparent objects *(right)*.

The contribution of this surface point to the object can be summarised by the fraction of views in which it is visible, obtained by averaging this function over all viewing directions to get the **average point visibility** $APV_S(x)$ which measures the **importance** of the point x on an average image,

$$APV_S(x) := \int_{\hat{n} \in \mathbb{D}} V_S(x, \hat{n})\, d\hat{n} \Big/ \int_{\hat{n} \in \mathbb{D}} d\hat{n} \tag{4.2}$$

where $\mathbb{D}$ is the set of all unit direction vectors in $\mathbb{R}^3$.

Conversely, the total visibility from a given direction can be expressed as the **projected surface area,** $PSA_S(\hat{n})$,

$$PSA_S(\hat{n}) := \int_{x \in S} V_S(x, \hat{n})\, dx \tag{4.3}$$

This is a measure of the size of an object when viewed orthogonally from a particular direction. While the point visibility and average point visibility functions can change discontinuously, the projected surface area always changes continuously.

The average taken over all viewing directions is the **average projected surface area** ($APSA$) and is the average size that the object would appear in an orthographically projected image.

$$APSA_S := \int_{\hat{n} \in \mathbb{D}} PSA_S(\hat{n})\, d\hat{n} \Big/ \int_{\hat{n} \in \mathbb{D}} d\hat{n} \tag{4.4}$$

$$= \int_{x \in S} APV_S(x)\, dx \tag{4.5}$$

For a closed convex object the APSA is always one quarter of its total surface area (Appendix A.2). It is convenient to define the **visible surface area** (*VSA*) as

$$VSA := 4 \times APSA \tag{4.6}$$

This is the effective surface area of the object — *i.e.* the analogue to its total surface area but accounting for self-occlusion — and is a view-independent measurement of the size of the object. The surface $\mathcal{S}$ can be replaced by the equivalent surface $\mathcal{S}'$ defined as

$$\mathcal{S}' := \{ \boldsymbol{x} : APV_{\mathcal{S}}(\boldsymbol{x}) > 0 \} \tag{4.7}$$

without altering its *VSA*.

This can be extended to semi-transparent volumes V by replacing the discrete visibility function by a continuous one, $V_{\mathcal{S}}(\boldsymbol{x}, \hat{\boldsymbol{n}}) \in [0, 1]$ indicating the contribution of point $\boldsymbol{x} \in V$ in direction $\hat{\boldsymbol{n}}$ (Figure 4.3 *(right)*).

4.2.2 Attribute Values

Property values are computed for an object by accumulating values and then averaging these results to get a single representative value for the entire object.

For a property $P(\boldsymbol{x})$, its average value over $\mathcal{S}$ can be defined as

$$\bar{P}_{\mathcal{S}} := \int_{\boldsymbol{x} \in \mathcal{S}} APV_{\mathcal{S}}(\boldsymbol{x})\, P(\boldsymbol{x})\, d\boldsymbol{x} \Big/ \int_{\boldsymbol{x} \in \mathcal{S}} APV_{\mathcal{S}}(\boldsymbol{x})\, d\boldsymbol{x} \tag{4.8}$$

$$= \int_{\boldsymbol{x} \in \mathcal{S}} APV_{\mathcal{S}}(\boldsymbol{x})\, P(\boldsymbol{x})\, d\boldsymbol{x} \Big/ APSA_{\mathcal{S}} \tag{4.9}$$

The property value is weighted by the visibility of each point at which it is calculated, so that the contribution of each point to the total property value is proportional to its importance.

The properties calculated using these methods can also be useful in situations other than the object replacement schemes described in this chapter. When calculating approximations, the visible surface area and in particular the average surface properties are of general interest.

4.2.3 Surface Traversal

The first approach to evaluating Equation 4.9 is to traverse the surface performing the integration. If the surface $\mathcal{S}$ is constructed from a set of disjoint surface elements $\mathcal{S}_i$ (Figure 4.4 *(left)*) so that

$$\mathcal{S} = \bigcup_i \mathcal{S}_i \text{ and } \mathcal{S}_i \cap \mathcal{S}_j = \varnothing \text{ for } i \neq j \tag{4.10}$$

then defining

$$P_{\mathcal{S}_i} := \int_{x \in \mathcal{S}_i} APV_{\mathcal{S}}(x)\, P(x)\, dx \tag{4.11}$$

the average value of a property P over $\mathcal{S}$ can be expressed as

$$\bar{P}_{\mathcal{S}} = \sum_i P_{\mathcal{S}_i} \Big/ APSA_{\mathcal{S}} \tag{4.12}$$

In order to simplify the calculation, Equation 4.11 can be approximated by removing the visibility from the integration, so that

$$P_{\mathcal{S}_i} \approx APV_{\mathcal{S}}(\mathcal{S}_i) \int_{x \in \mathcal{S}_i} P(x)\, dx \tag{4.13}$$

where average visibility is extended to an element of $\mathcal{S}$ as

$$APV_{\mathcal{S}}(\mathcal{S}_i) := \int_{x \in \mathcal{S}} APV_{\mathcal{S}}(x)\, dx \tag{4.14}$$

This is equivalent to assuming that the element has uniform visibility across its surface(Figure 4.4 *(centre)*).

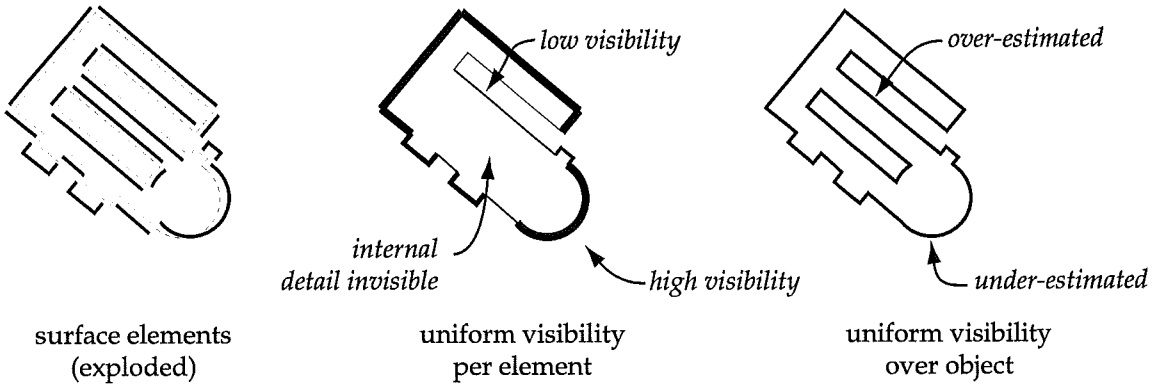


Figure 4.4: An object decomposed into elements shown in 2D *(left)*, the effect of assuming uniform visibility for each element *(centre)* and ignoring the visibility weighting *(right)*.

Polygons or triangles are typical surface elements, although it is not always necessary to break the surface down this far. Attributes for some objects, such as spheres or boxes can pre-calculated and rapidly substituted into Equation 4.13.

In practice, the visibility of an element is the difficult part to calculate and will often be estimated. The simplest such estimate is to assume that there is no self-occlusion and that

the element has a visibility of 1. This results in the property value

$$P_{\mathcal{S}_i} \approx \int_{\boldsymbol{x} \in \mathcal{S}_i} P(\boldsymbol{x})\, d\boldsymbol{x} \tag{4.15}$$

With this assumption, objects with a significant amount of internal detail will be poorly analysed, since this detail will be over-estimated in the results (Figure 4.4 *(right)*). In practice it is simpler to make this assumption and then use a different method in those cases where a significant amount of internal detail is suspected.

4.2.4 Object Sampling

Object sampling is a different approach to evaluating Equation 4.9, based on expanding it as

$$\bar{P}_{\mathcal{S}} = \int_{\boldsymbol{x} \in \mathcal{S}} \int_{\hat{\boldsymbol{n}} \in \mathbb{D}} P(\boldsymbol{x})\, V_{\mathcal{S}}(\boldsymbol{x}, \hat{\boldsymbol{n}})\, d\hat{\boldsymbol{n}}\, d\boldsymbol{x} \Big/ APSA_{\mathcal{S}} \tag{4.16}$$

This double integral can be evaluated approximately by random sampling. A simple visible surface ray-tracer is used to sample the object with random rays, and the attribute value accumulated at each point sampled.

Random rays are generated that are guaranteed to hit some convex bounding volume $\mathcal{B}$ of $\mathcal{S}$. The probability of such a ray hitting the object is **[Goldsmith87]**

$$VSA_{\mathcal{S}} \;/\; VSA_{\mathcal{B}} \tag{4.17}$$

so if m out of n rays intersect the object then

$$VSA_{\mathcal{S}} \approx VSA_{\mathcal{B}}\, m/n \tag{4.18}$$

Intuitively, larger completely visible surfaces are more likely to be hit by a random ray than smaller or partially occluded surfaces. The probability of a ray $\mathcal{R}_i$ hitting a point $\boldsymbol{x}_i$ on $\mathcal{S}$ is proportional to $APV_{\mathcal{S}}(\boldsymbol{x}_i)$, and so the average property value can be estimated as

$$\bar{P}_{\mathcal{S}} = \sum_{i}^{m} P(\boldsymbol{x}_i) \;/\; m \tag{4.19}$$

The effect of this sampling is to replace the original geometry (Figure 4.5 *(left)*), by a series of points (Figure 4.5 *(right)*), and suffers from the problems associated with sampling, such as aliasing.

Obtaining a good sampling rate for a complex object may make this approach prohibitively expensive. The tighter the reference bounding volume is to the object, the more efficient

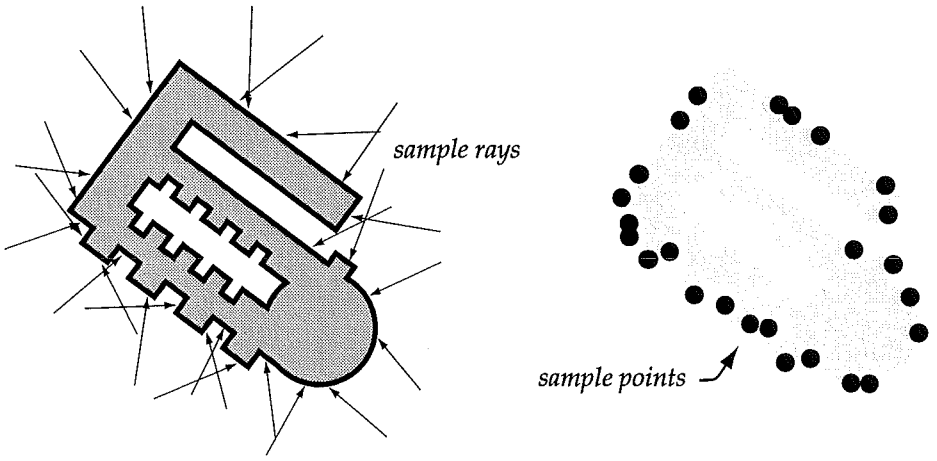


Figure 4.5: Calculating attributes on an object *(left)* by random sampling effectively replaces it by set of sample points *(right)*.

this method is, since fewer rays miss altogether. For some attributes, such as visible surface area, a fairly sparse sampling may be sufficient to obtain a reasonably accurate result due to the relative coherence of visibility. It may be possible to restrict the need for sampling by using a faster method first, such as the surface traversal described above, and then only performing progressively heavier sampling where there is an indication that a vastly different result might be obtained.

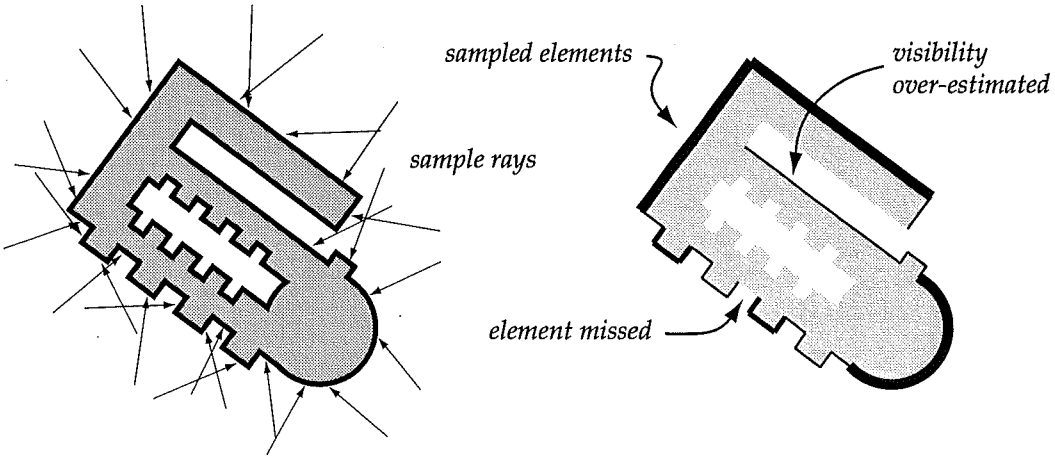


Figure 4.6: Sampling surface elements rather than surface points can reduce the sampling rate required *(left)*, but may introduce errors due to unrepresentative visibility results *(right)*.

One variation of the sampling strategy is to sample Equation 4.12 rather than Equation 4.9. Each sample ray in this situation picks out an *element* of S instead of a single point on S.

The advantage is that if the property value for the elements can be calculated efficiently, then the random sampling takes account of the visibility aspect of the computation. The disadvantage is that visibility is now only calculated on a per-element rather than a per-point basis (Figure 4.6). With this approach, however, a much lower sampling rate would be sufficient.

4.2.5 Location and Size

A representative location for an object could be computed in a number of ways. The current implementation uses the centre of mass of the object's surface since it is straightforward to compute in conjunction with the surface area. It is determined as the weighted sum of the centres of mass of the surface elements or sample points (depending on which approach is being followed).

An estimate of the visible surface area is used for the size of the object. This is arrived at through either a (possibly weighted) sum of the surface areas of elements or, when using a sampling approach, by using Equation 4.18.

There is one extension that is of practical use for surface traversal. This involves noting that the visible surface area can be bounded above by the surface area of any convex bounding volume $\mathcal{B}$ (Figure 4.7 *(left)*) so that $VSA_{\mathcal{B}}$ can be used to limit the estimate of VSA_S. The convex hull is the tightest possible convex bounding volume and thus would yield the best bound from this approach. As can be seen from Figure 4.7 *(centre)* and *(right)*, this does not necessarily provide a good approximation to the VSA; an over-estimation is quite likely to arise when approximating groups of objects.

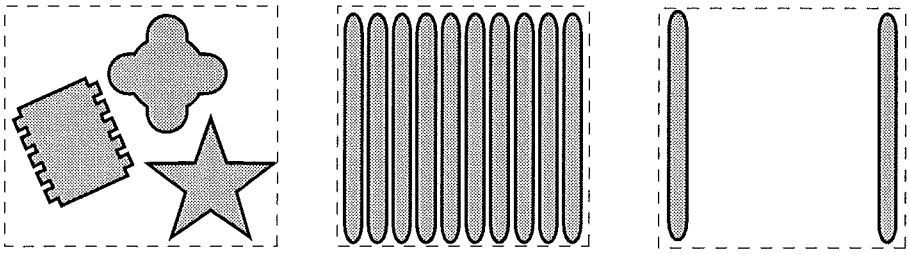


Figure 4.7: Calculating visible surface area using bounding volumes to limit surface area *(left)* which works well for dense groups *(centre)* but can over-estimate sparse groups *(right)*.

Differences between the two attribute gathering approaches are shown in Figure 4.8 where two books are intentionally modelled with a large number of pages which contribute a large invisible surface area. Their effect can be limited by the bounding volume shown in wire-frame *(left)* but, because it is not a very tight fit, this still over-estimates the VSA *(centre)*. If a sampling process is used to calculate the size of the box a much better result is obtained *(right)*.

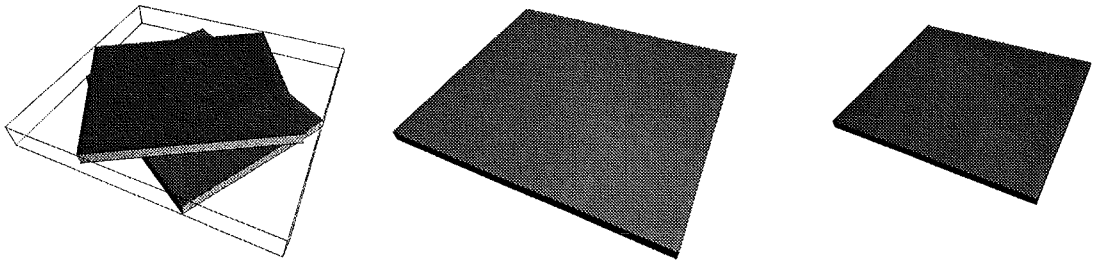


Figure 4.8: Two rotated books with axis aligned bounding volume *(left)*, replacement corrected by bounding volume area *(centre)*, and by sampling 200 times *(right)*.

4.2.6 Orientation and Aspect Ratio

Perceived orientation can be triggered by many visual cues, such as axes of symmetry, mass distribution, or even previous personal experience **[Biederman85]**. The brain is adept at identifying a **major axis** on a 2D outline (Figure 4.9) and can also do this for 3D shapes. Minor axes, passing through the centre of the object and perpendicular to the major axis will also be perceived, and in 3D the mutually perpendicular set of three axes defines an orientation for the object.

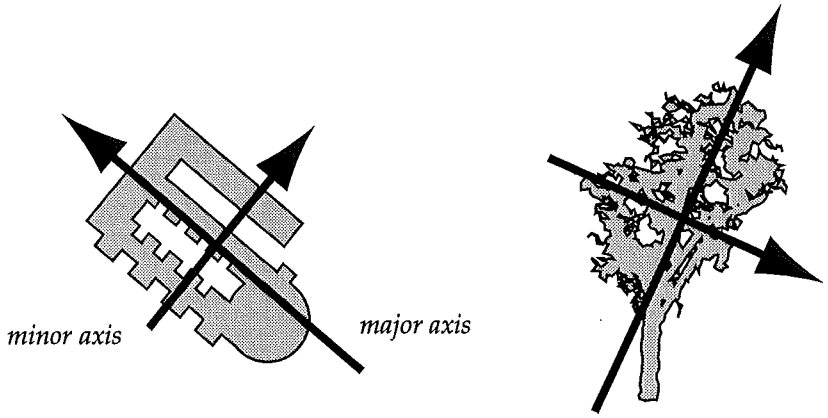


Figure 4.9: A two-dimensional image of an object and an example of how it is usually possible to identify natural major and minor axes.

One means of computing such an orientation is to calculate and examine the **surface inertia tensor** of an object, the surface analogue of the usual volume-based inertia tensor **[Goldstein50]**. It represents the second moments of the object surface about the centre of mass. Expressed as a matrix with respect to a coordinate system (x, y, z) with its origin at

the object's centre of mass,

$$\boldsymbol{T} = \int_{\mathcal{S}} \begin{pmatrix} y^2+z^2 & -xy & -zx \\ -xy & z^2+x^2 & -yz \\ -zx & -yz & x^2+y^2 \end{pmatrix} d\mathcal{S} \tag{4.20}$$

Since $\boldsymbol{T}$ is symmetric ($T_{ij} = T_{ji}$) it has just 6 independent real components, and is **self-adjoint** or **Hermitian**. This means that there will always be at least one coordinate system, consisting of three mutually perpendicular axes, with respect to which T_{ij} is **diagonal**,

$$\boldsymbol{T} = \begin{pmatrix} T_{00} & 0 & 0 \\ 0 & T_{11} & 0 \\ 0 & 0 & T_{22} \end{pmatrix} \tag{4.21}$$

The three diagonal elements, T_{00}, T_{11}, and T_2 are then called the **principal moments** of the object, and this coordinate system defines the **principal axes** of the object.

These principal axes can be identified by solving an eigenvalue equation.

$$(\boldsymbol{T} - \lambda \boldsymbol{I}) \, \boldsymbol{e} = \boldsymbol{0} \tag{4.22}$$

where $\boldsymbol{I}$ is the identity matrix. The eigenvectors $\boldsymbol{e}_i$ and eigenvalues λ_i are the solutions of this equation, the eigenvectors defining the coordinate system in which $\boldsymbol{T}$ will be diagonalised, where the diagonal values are given by the λ_i. Equation 4.22 can be solved numerically for a given inertia tensor $\boldsymbol{T}$ to find the principal axes and moments of inertia by using fast routines such as Jacobi Transformations [**Press92**].

The inertia tensor also describes the relative mass distribution of the surface — the **variance** of the surface along each axis — an indication of the aspect ratio of the object:

$$\begin{aligned} \int_{\mathcal{S}} x^2 \, d\mathcal{S} &= (T_{22} + T_{11} - T_{00})/2 \\ \int_{\mathcal{S}} y^2 \, d\mathcal{S} &= (T_{00} + T_{22} - T_{11})/2 \\ \int_{\mathcal{S}} z^2 \, d\mathcal{S} &= (T_{11} + T_{00} - T_{22})/2 \end{aligned} \tag{4.23}$$

These variances are actually used without further processing in the replacement schemes in the following section.

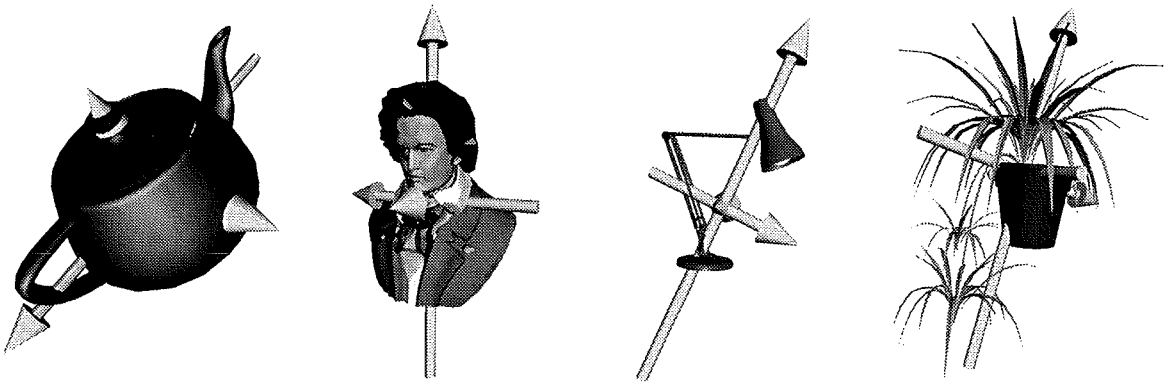


Figure 4.10: Original objects and principal axes calculated from surface inertia tensors — the lengths of the axes indicate the magnitude of the mass distribution along the corresponding axis.

4.2.7 Surface Tensor Calculation

During surface traversal, it is necessary to calculate inertia tensors for surface elements $\mathcal{S}$, and also to be able to accumulate the tensors for the different elements. Tensors can be added component-wise as long as they are calculated about the same centre point. The following transformation rule can be applied to translate a tensor representing objects of total mass M, from its centre of mass $\boldsymbol{c}$ to a new centre $\boldsymbol{p} = \boldsymbol{c} + \boldsymbol{t}$

$$T'_{ij} = T_{ij} + MP_{ij}(\boldsymbol{t}) \tag{4.24}$$

$$\text{where } \boldsymbol{P}(\boldsymbol{t}) = \begin{pmatrix} t_y^2 + t_z^2 & -t_x t_y & -t_z t_x \\ -t_x t_y & t_z^2 + t_x^2 & -t_y t_z \\ -t_z t_x & -t_y t_z & t_x^2 + t_y^2 \end{pmatrix} \tag{4.25}$$

Transformation by a rotation matrix $\boldsymbol{R}$ can be achieved by

$$T'_{ij} = \sum_{k}^{3} \sum_{l}^{3} R_{ik}\, R_{jl}\, T_{kl} \tag{4.26}$$

and uniform scaling by s as

$$T'_{ij} = s^2\, T_{ij} \tag{4.27}$$

The restriction to *uniform* scaling is due to the integration over the object's surface rather than its volume — in general surface area cannot be scaled non-uniformly without recomputation. In cases of non-uniform scaling, surface elements need to be transformed *before* the calculation of the surface tensor can be performed.

The properties for a surface can be combined from the values for the surface elements using these rules. For elements with properties $\{\ A_1, \boldsymbol{c}_1, \boldsymbol{T}_1\ \}$ and $\{\ A_2, \boldsymbol{c}_2, \boldsymbol{T}_2\ \}$, the combined

surface has properties $\{ A_3, \boldsymbol{c}_3, \boldsymbol{T}_3 \}$ where

$$\begin{aligned} \text{surface area} \quad & A_3 = A_1 + A_2 && (4.28) \\ \text{centre of mass} \quad & \boldsymbol{c}_3 = (A_1\boldsymbol{c}_1 + A_2\boldsymbol{c}_2)/(A_1 + A_2) && (4.29) \\ \text{surface tensor} \quad & \boldsymbol{T}_3 = \boldsymbol{T}_1 + \boldsymbol{T}_2 + A_1\boldsymbol{P}(\boldsymbol{c}_3 - \boldsymbol{c}_1) + A_2\boldsymbol{P}(\boldsymbol{c}_3 - \boldsymbol{c}_2) && (4.30) \end{aligned}$$

For primitive shapes such as boxes, spheres, cylinders, triangles the inertia tensor can be rapidly computed from a stored parametrised form, and then transformed appropriately. Appendix A.1 lists inertia tensors and other properties for some common primitives.

4.2.8 Surface Properties

In simplification schemes, an object usually only contains surfaces with a single set of surface properties. Replacement schemes are much more likely to deal with compound objects which contain several sets of properties for different parts of the object. A simple average set of surface properties is obtained by accumulating these properties over the surface of an object. More complex ways of dealing with surface attributes are covered in Section 4.4.

The visibility weighting can be quite important. For example in Figure 4.8 on page 83 the white pages of the books have a large total surface area, so the surface-based analysis *(centre)* produces a colouring with very little red content from the cover. The sampling approach, however, by including the variation in visibility, improves this greatly *(right)*.

Since the replacement object, with the average set of surface properties, is passed through the current shading model, the object still responds to local lighting effects. The simplified geometry will therefore also affect the appearance of the replacement, with the distribution of surface normals playing an important part in calculating the overall colour. Preserving the distribution of normals to some extent would be beneficial, but this places too many constraints on the simple replacements considered below. One approach attempts to address this within a polyhedral approximation scheme [**Brechner95**], but it remains difficult in general to both simplify *and* approximate distributions of normals.

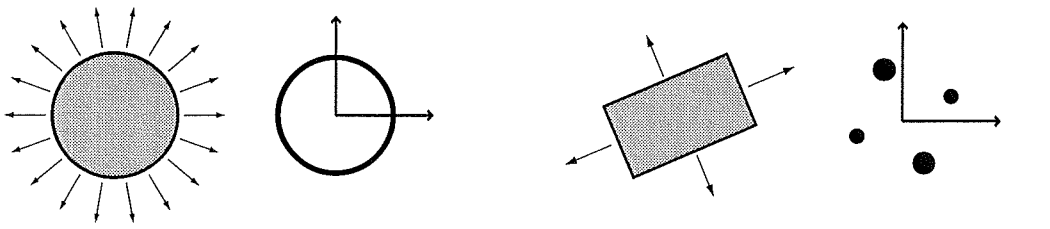


Figure 4.11: The distributions of normals for a sphere and a box.

It would be possible to capture distributions of normals by applying an inertia tensor approach, but using a pseudo-surface formed by the normal vectors instead of the object surface itself (Figure 4.11)

$$\boldsymbol{N} = \int_{\mathcal{S}} \begin{pmatrix} n_y^2 + n_z^2 & -n_x n_y & -n_x n_z \\ -n_x n_y & n_x^2 + n_z^2 & -n_y n_z \\ -n_x n_z & -n_y n_z & n_x^2 + n_y^2 \end{pmatrix} d\mathcal{S} \tag{4.31}$$

Flat surface elements of area A and normal $\hat{n}$ thus reduce to a single point mass at position $\hat{n}$ and with mass A.

4.3 Calculating Replacements

Replacements are created by a **matching** process. This constructs approximations which, when analysed in the manner described in the previous section, have the same attribute values as the objects they replace.

Three categories of replacements have been experimented with:

- spherical approximations,
- directional approximations using either boxes or ellipsoids,
- transparent approximations which modify these replacements where appropriate.

4.3.1 Spherical Replacements

The simplest replacement is the sphere. This is placed so that the centre matches the original's location with the surface properties set to the average properties computed, and the radius r chosen so that the surface area matches its visible surface area

$$r = \sqrt{VSA/4\pi} \tag{4.32}$$

When the sphere *and* the objects it replaces appear small, the replacement works very well, since size and location are maintained. The sphere has a fixed spherically symmetric distribution of normals which may differ significantly from that of the original, causing noticeable differences in lighting values. As a coarse approximation, the sphere suffers from a number of problems, not least that it is difficult to tell one sphere from another in a room full of them. In an example system using spherical replacements at large scales, little information could be drawn from the mass of similarly sized spheres serving as approximations to the objects in the scene. For larger scale approximation, and also to represent the distribution of normals of objects more effectively, a replacement with more degrees of freedom is required.

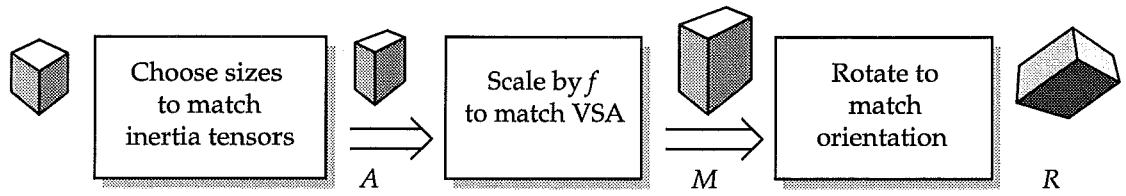
4.3.2 Box and Ellipsoid Replacements

By using a rotated box (Figure 4.12 *(left)*) as a replacement, orientation and aspect ratio can be matched as well as size, location and colour. The correct orientation is achieved by applying a rotation matrix to the box; the aspect ratio controls the relative lengths of the sides. The surface area of the replacement now varies with viewing direction in a way that usually reflects the variation of the original.

For a box replacement, the variances of the box along its principal axes can be expressed in terms of the lengths of the sides (Appendix A.1.3). Matching these to the variance attributes results in three simultaneous equations in three unknowns, which can be solved numerically for the dimensions of the box. A final scaling factor is applied so that the surface area of the result is equal to the visible surface area calculated for the original. Thus the original object O is used to generate an initial replacement A which has the same inertia tensor as O, and which is then scaled by a factor

$$f^2 = VSA_O / VSA_A \tag{4.33}$$

and rotated to obtain the final replacement R:



An alternative oriented replacement is an ellipsoid (Figure 4.12 *(right)*) — a sphere scaled unequally along the orientation axes (a simple modification to the orientation matrix). The relative magnitudes of the scaling factors are fixed from the aspect ratios of the object, and the absolute magnitude set so that surface areas match. Unfortunately, there is no closed-form expression for the surface area of an ellipsoid, but a numerical estimation is straightforward.

The main difference from oriented boxes is the curvature of the surface. While there are only six distinct normal directions for the box, the surface normals of the ellipsoid are distributed over all directions. These directional replacements cure many of the problems associated with spherical replacements. In many cases they follow much more closely the shape of the original, and also convey the rough shape of the object, making them much more suitable for large-scale approximations. While rendering ellipsoids is slower than drawing boxes on a conventional Z-buffering system, they are ideally suited for ray-tracers.

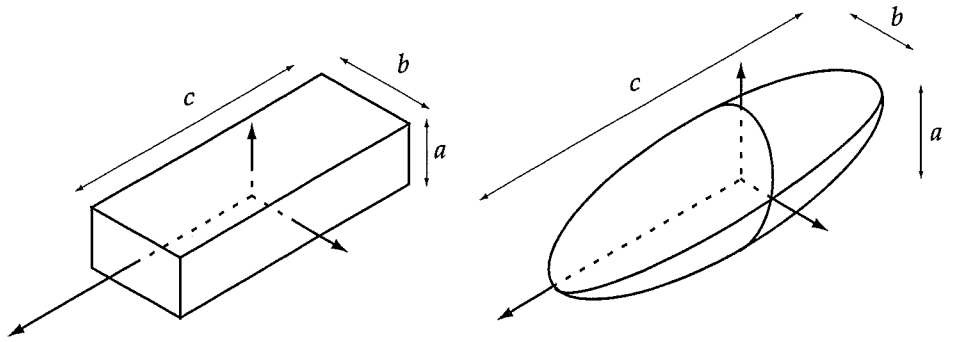


Figure 4.12: The parameters defining an oriented box *(left)* and ellipsoid *(right)*.

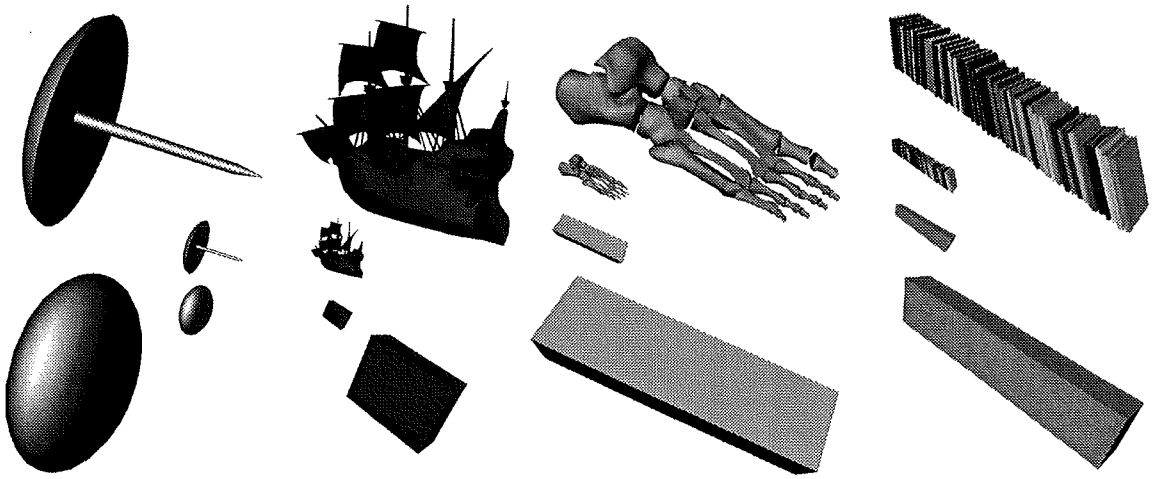


Figure 4.13: Four objects together with oriented replacements computed automatically using the methods described in this section shown at different sizes for comparison.

4.3.3 Partially Transparent Replacements

Objects which are widely separated from each other are not approximated very well by solid replacements (Figure 4.14), since the replacement will either be too large or far removed from the outer objects.

One solution is the use of partially transparent replacements, where the object is blended into the image with a controlled opacity. For instance, a flock of birds looks better if it is replaced by a transparent shape rather than a single opaque block, since the objects behind them remain partially visible.

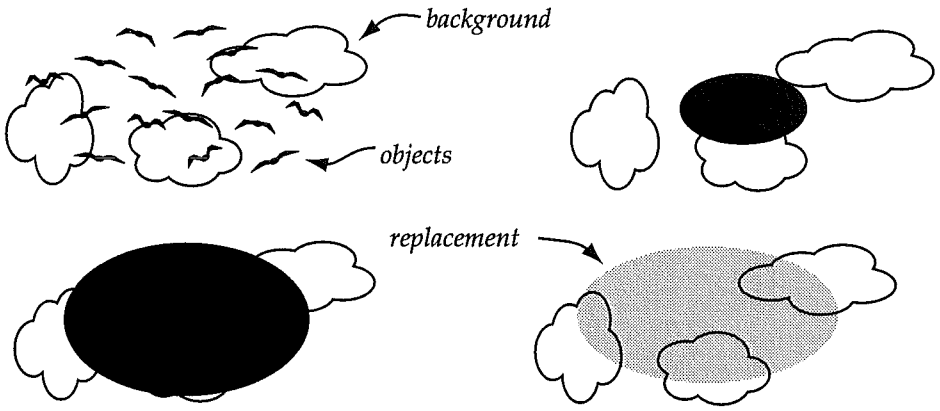
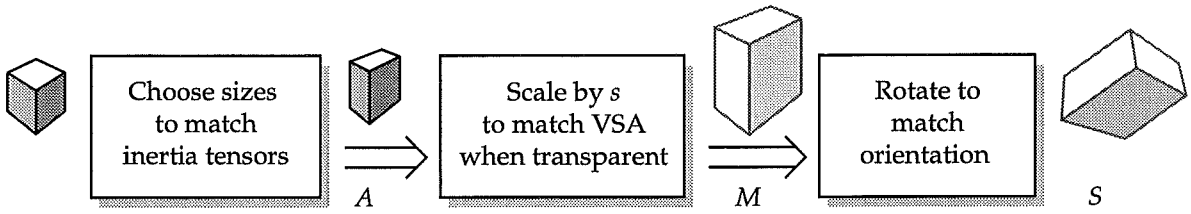


Figure 4.14: Approximations for sparse groups of objects *(top left)* — solid approximations with correct *(top right)* and incorrect sizes *(bottom left)* and partially transparent replacement *(bottom right)*.

The partially transparent replacement is modelled as a shell with constant opacity ρ, resulting in a uniform opacity across the image. The semi-transparent replacement is now obtained from the initial replacement A through scaling by a new factor s:



This added degree of freedom allows *both* the inertia tensor of S and its visible surface area to be matched to that of the original object. Since under uniform scaling area increases by the square of the scale factor and the inertia tensor by the fourth power of the scale factor, this relationship reduces to the following equations

$$\begin{aligned} VSA_O &= VSA_S \\ &= \rho\, s^2\, VSA_A \\ &= (\rho\, s^2/f^2)\, VSA_R \\ &= (\rho\, s^2/f^2)\, VSA_O \end{aligned} \tag{4.34}$$

$$\begin{aligned} \boldsymbol{T}_O &= \boldsymbol{T}_S \\ &= \rho\, s^4\, \boldsymbol{T}_A \\ &= \rho\, s^4\, \boldsymbol{T}_O \end{aligned} \tag{4.35}$$

which can be solved for ρ and s in terms of f

$$s = 1/f \tag{4.36}$$
$$\rho = f^4 \tag{4.37}$$

where f is given by Equation 4.33. If $f > 1$ this calculation produces a density greater than 1, indicating that transparent replacement would not be suitable. Figure 4.15 shows two examples of transparent replacements applied amongst other objects which avoid the problems of being too large or centralised and successfully approximate widely distributed objects. One rendering variation when ray-tracing such primitives is to use **stochastic transparency** — the transparency value is used directly as a probability of the ray hitting the solid surface as suggested in [**Thompson91**], avoiding having to continue to intersect rays with other primitives.

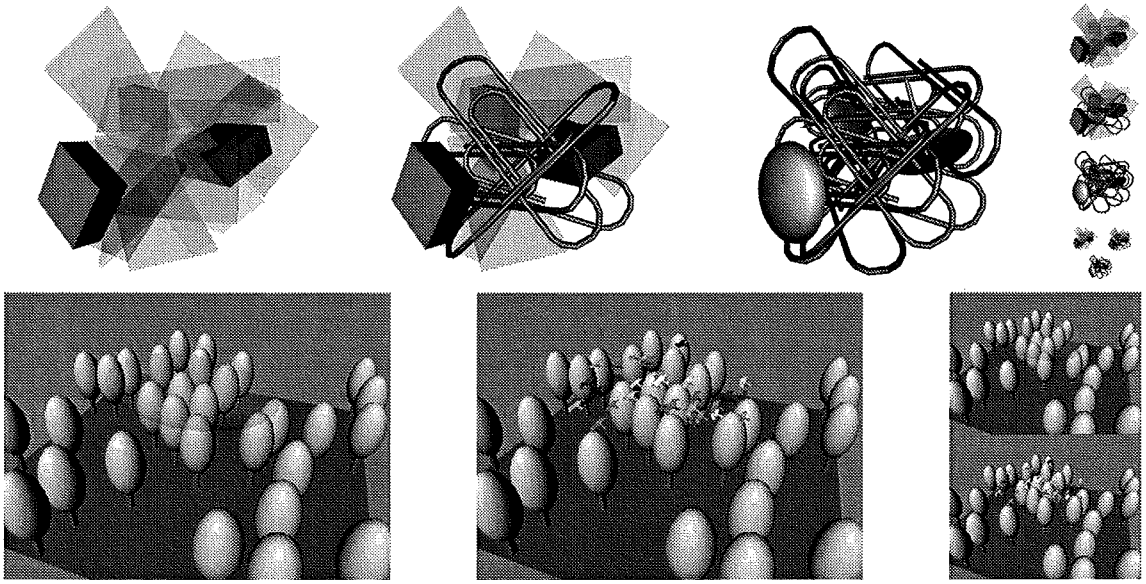


Figure 4.15: Transparent approximations applied to a collection of objects displayed at different sizes *(top)* and to a group of objects against a background environment *(bottom)*.

4.3.4 Implementation and Results

These replacement schemes have been implemented on top of Open Inventor [**Wernecke94**]. The surface traversal attribute gathering can be performed in a single pass through the dataset, and is $O(n)$ in the number of surface primitives since the assumption is made that all elements have equal visibility. Open Inventor provides a mechanism where all objects can be decomposed into triangles, and by default attributes

are calculated for these elements during surface traversal. Some primitives, such as spheres and boxes, are not decomposed, since expressions for their attributes can be computed directly. After traversal, the results are averaged, and then stored in a new 'Attribute' node to form a decorated scene graph. The surface traversal step completes in only one or two seconds, even for quite complex objects.

Object sampling is performed after the initial pass, with each ray requiring an $O(\log n)$ traversal of the database. A small number of samples is used to estimate whether denser sampling will be useful, and finally the results are merged into those obtained during surface traversal. The replacement algorithms then use the information in the 'Attribute' nodes to generate approximations.

The scheme has been successful on the test objects used, and some further results are shown in Figure 4.16. In particular, axes of symmetry, which provide an important clue for the observer, are identified, and the rough mass distribution is retained through the approximation, all independent of the coordinate system. The choice between cubical and ellipsoid replacements is currently specified manually, but could potentially be automated. It would be interesting to experiment with intermediate shapes as replacements, perhaps using superquadric surfaces which can blend between boxes and ellipsoids [**Barr92**]. Ellipsoid replacements are less efficient for simple objects in the context of a polygon-based Z-buffer rendering system due to the per-polygon overheads, but offer advantages for a ray-tracer.

One characteristic of the replacement objects is that they do not necessarily satisfy the same geometrical constraints as the original objects. So, for instance, a stack of books on a shelf might be replaced by a box that penetrates the shelf. This cannot, in general, be remedied without including surrounding geometry in the approximation process, or placing some constraints on the replacement during generation. It may be possible, for example, to constrain the orientation of the replacement to lie on a plane, and then solve for a replacement by minimising the errors. If the diagonalisation of the inertia tensor does not have a *unique* solution then an extra degree of freedom becomes available to the matching process. This case is easy to identify since it arises when two (or three) of the eigenvalues are equal — rotation in the plane perpendicular to the other eigenvector leaves the inertia tensor unchanged.

In general, this scheme works well for groups of *objects*, but less effectively for *environments*. This is because an environment, such as the walls of a room, possesses important properties such as enclosure, which are not identified in this process, and therefore can be violated in the replacement. Ideally, objects such as walls and rooms would be modelled at a higher level, where knowledge of the importance of such aspects can be taken account of in the approximation process.

Figure 4.16: Results from the replacement algorithms applied at different levels to three objects. The images show the full models, containing 45 000, 46 000 and 20 000 triangles respectively, *(left)*, two levels of approximation *(centre)*, and then these representations at smaller sizes *(right)*.

4.3.5 Other Replacement Schemes

Bounding volumes are often used as simple, easy to compute, replacements for groups of objects. While these are capable of describing the extent of objects, they have a number of

disadvantages. Firstly, by definition, the size of the replacement will be an over-estimate, often by a large factor. Secondly, orientation is usually restricted to the coordinate system in which the objects are constructed rather than a natural orientation of the object. Another characteristic is that the bounding volumes depend only on points of the object lying on its convex hull, and will thus be the same whatever the distribution of internal features.

Axis-aligned boxes are one of the most common bounding volumes in use. When applied as approximations, these result in objects appearing all to be aligned with each other and becoming hard to distinguish. Furthermore, the size of the bounding boxes will change depending on the orientation of the objects enclosed. A cylinder, for example, if nearly aligned with a coordinate axis, is approximated well, but if it is slanting then the approximation is much worse (Figure 4.17). A radiosity system using bounding boxes scaled after estimating the VSA of an object by a restricted sampling parallel to the coordinate axes is described in [**Rushmeier93**].

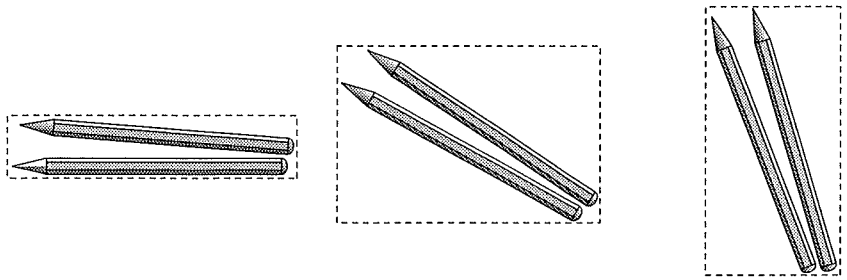


Figure 4.17: How axis-aligned bounding boxes change when their contents are rotated.

Bounding volumes are, however, easy to compute, and moreover often exist in hierarchies for other reasons. The use of polyhedral bounding volumes as approximations has been proposed recently by [**Brechner95**]. These are created from the intersection of the individual faces in a polygonal model after being translated as planes along their normal until all points in the object are behind the plane (Figure 4.18). The reason for this construction is that the replacement will preserve many of the surface normal directions of the original. While this is true in part, the relative proportion of the surface with each normal changes, and some faces are removed altogether. Once again, this scheme results in an over-estimation of object size. However, it *does* correct some of the orientation problems of the axis-aligned bounding box, although is not as simple to compute, and relies on a polygonal model as input. A further simplification of the polyhedron is also required in order to ensure that the approximation is actually simpler than the original.

At first glance, the convex hull appears to be a good approximation, but it is much harder to compute. A simpler bounding volume to calculate is based on "slabs" [**Kay86**] sandwiching the object between parallel planes. A set of seven slabs (one for each pair of opposite faces of a truncated cube) yields a convex volume with up to 14 faces which generally provides a tight bound for the convex hull of an object.

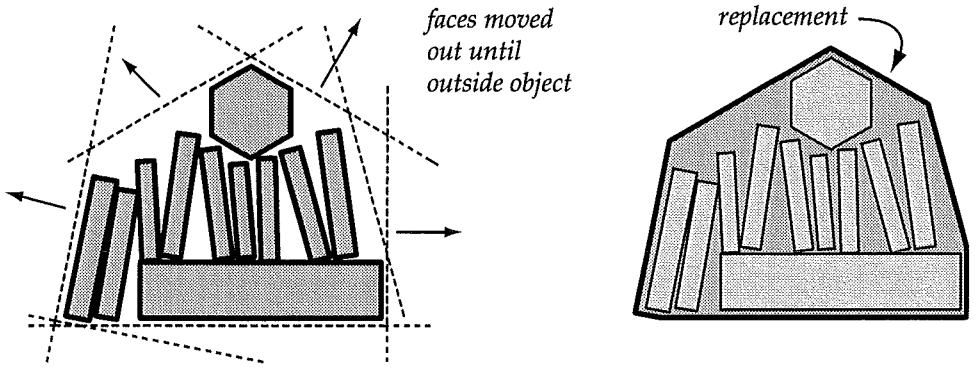


Figure 4.18: Calculating a bounding polyhedron by translating the faces of the input objects (only some faces shown).

The calculation and matching of attribute values offers a much more object-centred approach, and by extending the range of the attributes calculated should prove much more flexible for describing objects in high-level terms that are familiar to observers. The following sections demonstrate how these values can also be used in other aspects of the approximation process.

4.4 Surface Models

The identification of visible surfaces is only one stage in the rendering process. Objects in the model will also contain information to be used by the **illumination model** to calculate the colours that they will appear [**Foley90**]. The illumination model consists of three main parts (Figure 4.19):

- **lighting model** — the contribution each light source makes at a given point on an object, from a given direction,
- **surface model** — what colour is seen when light falls on a given point on an object,
- **shading model** — how pixel colours are generated from these results.

All of these models usually involve approximations, as they are an attempt to simulate the complex interaction of light energy with the physical environment at a hierarchy of levels [**Kajiya85b**]. Some are based on physical observation and theoretical results, while others have been developed for their appearance and computational advantage.

Lighting and shading models are discussed in Chapter 5 — the current chapter examines surface models since they are associated with objects, independently from rendering. First, a general framework is presented within which surface models are described and used. This includes the use of surface-maps to give the impression of geometric detail on the

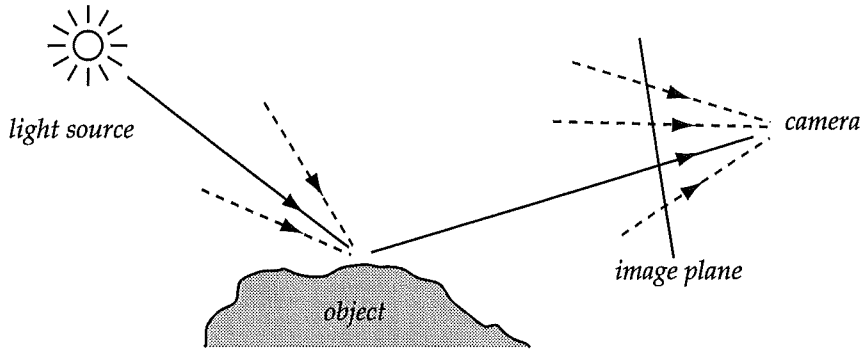


Figure 4.19: The relationship between the three components of an illumination model.

surface of objects. The second part of this section describes a class of object replacements, **image-map objects**, based upon surface-maps.

4.4.1 Surface Descriptions

The interaction of a surface with incoming light is often described in terms of the **bi-directional reflection distribution function** (BRDF) **[Foley90]**, $R(\lambda; \phi_i, \theta_i; \phi_v, \theta_v)$, which gives the light emitted in direction (ϕ_v, θ_v) in terms of the light incident from direction (ϕ_i, θ_i) and the wavelength λ (Figure 4.20). Surface models describe approximations to this function for all points on a surface.

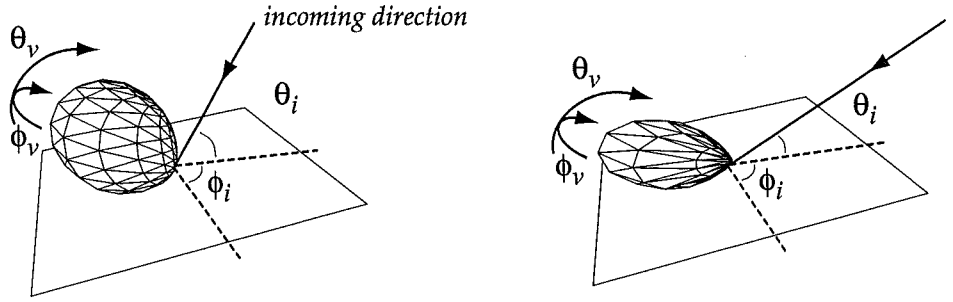


Figure 4.20: An illustration of the bidirectional reflection distribution function (BRDF) for a surface showing the response for two incoming directions.

The most widely adopted approximation is the **Phong model [Phong75]**, which simplifies the BRDF into ambient, diffuse and specular components, describing the surface in terms of these coefficients and a specular parameter which controls the simulated roughness of

the surface. Other models include the **Cook and Torrance model** [**Cook82**] which is based on an energy transfer model using a micro-facet description of rough surfaces.

One of the most effective enhancements for adding detail to scenes is to *vary* the surface model over the surface concerned. Most mechanisms for achieving this can be described by a **surface-map** which enables the surface description to be retrieved at any point on the surface. The map may be implemented by looking up values from an array of samples, evaluating a mathematical expression or even executing a function. The most common of these is the array of values, since the others can be converted to this form, after which the values require little further processing.

Some different names and uses for these generic maps include:

- **Texture-Mapping** pre-defined arrays of pixels modifying surface properties, in particular the base colour of the surface.
- **Bump-Mapping** arrays of vectors modifying the surface normal to simulate bumpy surfaces.
- **Displacement-Mapping** arrays of vectors altering the actual surface location; a method only suitable for very flexibly defined renderers.

While texture-maps can reduce complex geometry in favour of a complex surface-map, there is a penalty to pay in the storage required. A real-time hardware graphics system, for example, may require very fast access to a large number of surface-maps which need to be placed in a limited amount of special storage ([**Akeley93**] [**Maciel95b**]).

4.4.2 Approximation and Surface Models

Complex illumination models applied to geometrically simple surfaces aim to give the impression of a more complex surface without the overheads of modelling it, and are as such already an approximation.

The use of different combinations of geometry and surface model enable an appropriate overall quality to be chosen for the object. For example, a bump-mapped, textured surface, illuminated by area light sources, is not needed for an object a few pixels high. Conversely, a texture-mapped surface when viewed closely appears to be a coloured but flat image — the illusion of detail created by this process has broken down.

As with geometry, surface models can either be simplified or replaced. Replacement in this case can either be over the same geometry, or include the generation of a surface description over one object that approximates a different surface description on a *different* object (Figure 4.21). Automatic techniques are again necessary to produce approximations whose quality is linked to that of the geometric approximation. While some work has been done integrating different surface models for a single surface, the general case is much more complex.

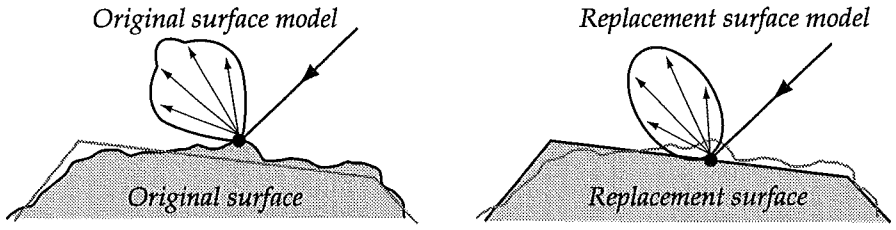


Figure 4.21: Replacing both the surface description and the surface geometry simultaneously.

Data-structures have been proposed for storing BRDFs in terms of spherical harmonics [**Cabral87**] [**Sillion91**], stratified spheres [**Becker93**] [**Gondek94**], or wavelets [**Schröder95**] [**Lounsberry94**], each of which could be used as the basis of a multi-resolution storage mechanism for BRDFs. **Shade trees** [**Cook84**] are a flexible means of specifying a wide variety of surface models (as well as the other aspects of the illumination model). The RenderMan system [**Upstill90**] extends these to a general notion of procedural **shaders**. This structure would be another interesting subject for approximation; replacing subtrees by simplified shading nodes would lead to multiple representations for surface properties.

Surface-maps stored as arrays of values are immediate candidates for simplification. For example, when placing images taken from photographs of the real world onto surfaces in a synthetic world, the photo texture-maps should be accessible at different resolutions since they may be used at different sizes on an image. A common technique for achieving this is MIP-mapping [**Williams83**], but other techniques from 2D image storage and access, such as wavelet image representation [**Sweldens94**], are also applicable.

Methods for correlating *different* surface models are described in [**Becker93**] [**Cabral87**] and [**Westin92**] showing how one can be *replaced* by another so that these models blend smoothly. The first concentrates on maintaining local averages through the transitions from displacement-mapping to a modified version of bump-mapping and from this to BRDFs described over the same surface. The second calculates BRDFs from bump-maps, while the last demonstrates how BRDFs can be computed to represent more detailed regular geometry, such as cloth, over a smooth surface by bombarding a sample with rays to calculate average properties. These methods could be integrated as extensions of the attribute gathering and matching techniques introduced in this chapter to more detailed properties of the surface model.

When approximating groups of objects by a single object it is the local variation (*i.e.* the detail) that is often lost; surface-mapping techniques are an ideal way of including such information cheaply. Standard texture-mapping is one example, but the use of bump-mapping or displacement-mapping to indicate the presence of detail remain under-investigated. The surface-maps that are considered in this section are based on pre-computed arrays of values. There may also be scope for filtering out a functional or statistical description of the surface variation, as has recently been proposed for 2D textures [**Heeger95**]. The next sec-

tion, however, examines how texture-maps can be generated automatically to increase the quality of approximate objects.

4.4.3 Surface Models for Approximating Geometry

In order to calculate the values at a point in a surface-map over a surface S, it is necessary to examine which parts of the original surface T influence that point and to what degree. Ideally it would be possible to establish a **projection** $\phi : S \rightarrow T$ that defines where on T each point on S obtains its value from. Unfortunately, the projection is not so simple, since it should take into account, when considering all viewing directions, that many points on T might be seen through a single point on S (Figure 4.22) — an ideal projection would take into account the probability of each viewing situation and also possible self-occlusion within T.

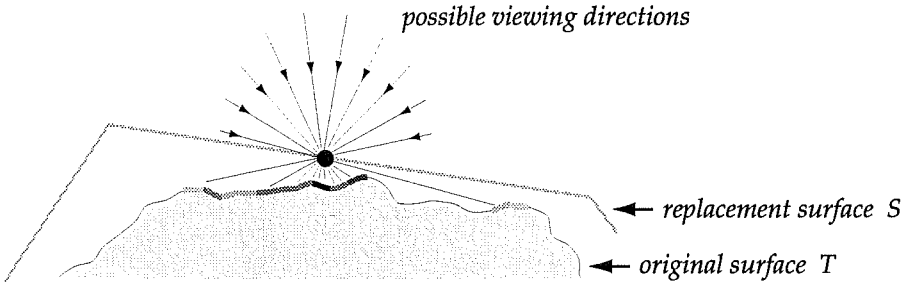


Figure 4.22: Projections between surfaces taking into account possible viewing scenarios and their likelihood, with the original surface shaded according to its contribution to the point shown.

When the surfaces have very similar geometry, a simplified one-to-one projection *can* be used. The assumptions that such projections are based upon, however, do not necessarily hold as the surfaces diverge from each other. The two most practical projections (Figure 4.23) are:

- **Central Projection** $\phi(\boldsymbol{y})$ is the outermost intersection point of a ray through $\boldsymbol{y}$ from a central point $\boldsymbol{c}$ with the surface T;
- **Orthogonal Projection** $\phi(\boldsymbol{y})$ is the outermost intersection point of a ray through $\boldsymbol{y}$ perpendicular to S with the surface T.

Nearest-point projection (Figure 4.23 *(right)*) is less practical, but might be considered a sensible choice as well. All of these projections will not produce a result if the rays do not intersect T. Such points need to be marked as **transparent**, and this information should be included in the surface-maps.

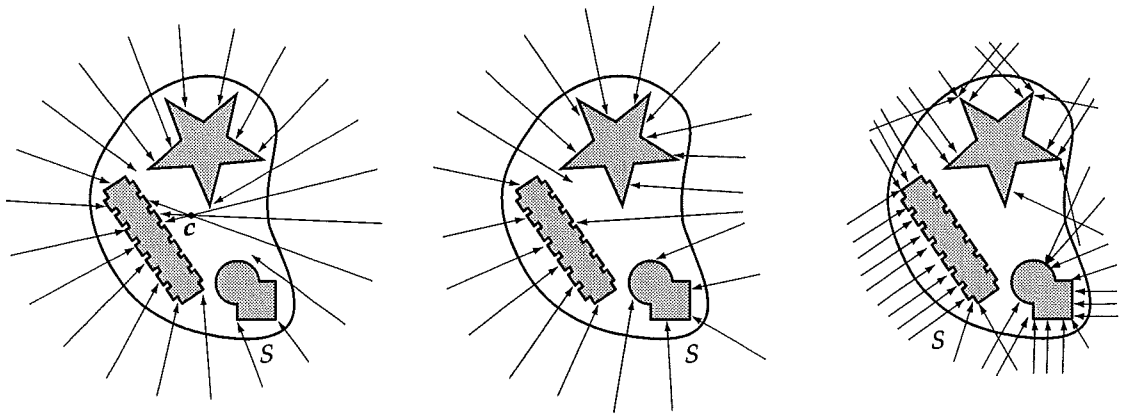


Figure 4.23: Different projections between between two surfaces — central projection *(left)*, projection orthogonal to the replacement surface *(centre)*, and nearest point projection *(right)*.

4.4.4 Example – Texture-Mapped Approximations

Texture-mapping is a common feature of rendering systems and usually modifies or replaces the diffuse surface component in a Phong surface model, but may also be used in conjunction with other components of different surface models. While they are usually constructed manually, texture-maps can be generated *automatically* from higher quality representations.

The simplified geometry can be generated either by the replacement methods given earlier in this chapter, or from a bounding volume [**Maciel95b**], or simply from a manually-constructed approximation. A texture-map coordinate system needs to be established over the surface elements forming the approximation and a resolution needs to be determined.

Automatic texture-map generation has been incorporated into the replacement scheme described in Section 4.3.4 to calculate textures that can be included on the faces of box replacements. Central projection techniques were found to be problematic when the centre of projection falls inside some part of the original object since no part of the texture is transparent. The distortions introduced when mapping to a non-spherical replacement are noticeable to the eye if there are many straight edges in the original objects. Furthermore, non-standard projections require a method such as ray-tracing to sample the objects. Orthogonal projection can take advantage of standard rendering algorithms to generate the textures, although surface information cannot be retrieved using this approach.

The example system uses the Z-buffering hardware to create textures under uniform lighting conditions which are then mapped onto diffuse white surfaces. Transparency can be identified from the pixels which remain at the background intensity. In general, a separate transparency-map can be generated indicating the average transparency for each pixel.

Since a replacement with transparent areas will appear to have a smaller *VSA* than a corresponding opaque one, when including transparency, the replacement should be larger. Using a texture-mapped bounding box, for example, will produce a replacement with a good approximation to the *VSA* of the original object. The disadvantage is that the texture planes are moved further from the centre of the object resulting in more distortion. The next section proposes a modified object geometry that lessens this effect.

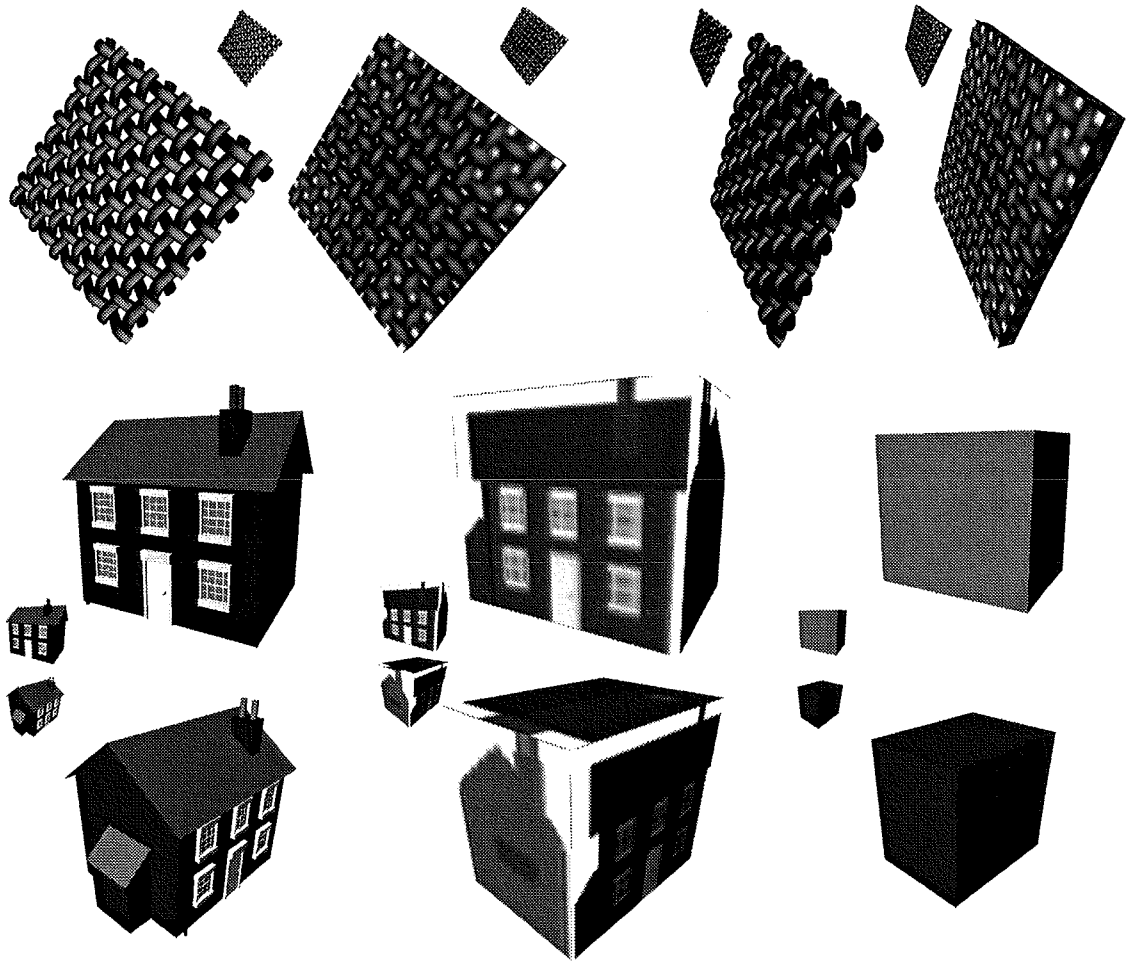


Figure 4.24: Examples of objects together with bounding boxes covered with automatically generated texture-maps of about 50 texels across.

Figure 4.24 *(top)* shows a patch of cloth modelled as strands of material effectively approximated by a texture-map, although slight differences can be seen at oblique angles where a more detailed surface model would be required. The lower images show a house with its bounding box texture-mapped — from a corner view, there are obvious problems with being able to see through the middle of the object. The lack of local lighting effects is also

visible on the image of the roof. The last image shows the use of this technique to generate box approximations with a *single* colour per-face rather than having the same material properties for the whole box. The creation of each texture mapped object takes less than a minute in total and results in an object of constant complexity, whatever the input.

Texture-mapped bounding boxes, obtained by rendering images in advance, are used as replacements by [**Maciel95b**], relying on using a similar lighting situation when generating texture-maps as when rendering them. To avoid lighting-dependency, separate maps can be generated for different components of the surface model, including transparency. For a complex surface model this may result in a large volume of texture information; consequently, a simple surface model is preferable with perhaps a colour, shininess and transparency-map requiring only five values per pixel, while the remaining components are modelled with a single value for the whole surface. This leads towards a surface model where each component is stored and used at different resolutions depending on the variation in its values.

4.4.5 Example – Image-Map Objects

This section extends texture-mapped objects to a class of less physical approximating objects that consist entirely of images, some of which are chosen for display based on the viewpoint.

The starting point is a well-known technique for texture-mapping objects that are rotationally symmetric. A single texture-mapped polygon, called a **billboard** in IRIS Performer [**Rohlf94**], rotates around the axis of symmetry so that it always faces the viewer (Figure 4.25). These are fast to render, with the apparent normal n to the object seen from direction v being

$$n = v - (v\ \hat{a})\ \hat{a} \tag{4.38}$$

where $\hat{a}$ is the axis of symmetry of the object. Commonly used as approximations to trees, the method can also be applied to furniture or even people.

Unfortunately, since the billboard always appears to rotate about a fixed axis, the object almost disappears when viewed nearly along this axis. One solution is to include another texture in the plane perpendicular to the axis of rotation representing the "top view" of the object. This is generalised here to a number of intersecting, but stationary, textures approximating an object, each passing through the centre of the object (Figure 4.26). These provide a better description of asymmetrical objects; three mutually perpendicular textures, for instance, will always give a good impression of solidity from any viewpoint.

Figure 4.27 indicates the problems that can occur with both textured bounding box and intersecting texture schemes. Since a point on the original object can be projected onto several of the textured faces of the replacement, when viewed diagonally it will appear at up to three points simultaneously. This occurs mainly for points on the object that are far away

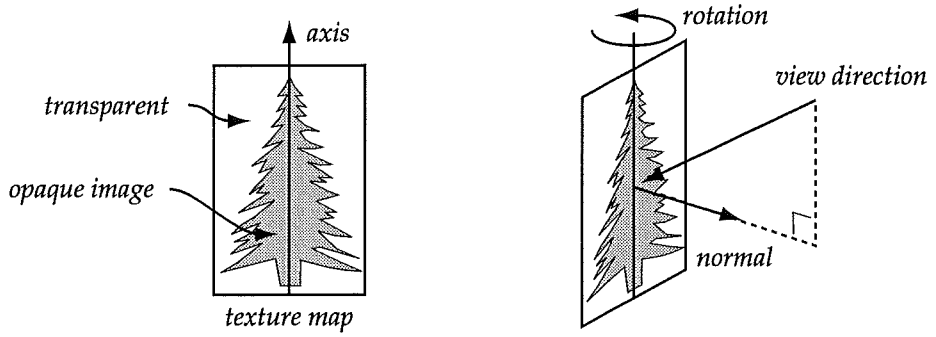


Figure 4.25: Billboard primitive and viewing geometry.

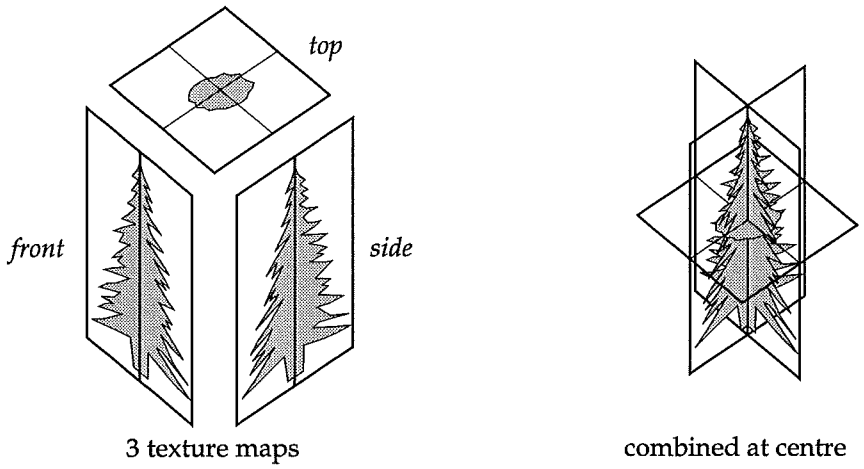


Figure 4.26: An object formed from three intersecting textures.

from the texture planes. By placing the planes between these two extremes the errors can be minimised (Figure 4.27 *(right)*). The sides of a box replacement, calculated in Section 4.3.2, defines suitable positions to place the images, reflecting the distribution of the surfaces of the object.

The images in Figure 4.28 allow these techniques to be compared, showing how the distortion is minimised by placing the texture planes on the edges of the box approximations. This method has the advantage that it will reduce to the bounding box method where appropriate, *i.e.* for objects whose surfaces are predominantly near sides of the bounding box.

The final scheme further reduces the effect of multiple images by dynamically choosing one or more images from n depending on the viewpoint. Choice is based on the angle between the normal to each image $\hat{n}_i$ and the view direction $\hat{v}$ by calculating

$$c_i = \hat{n}_i \, \hat{v} \quad \text{for } i = 1 \text{ to } n. \tag{4.39}$$

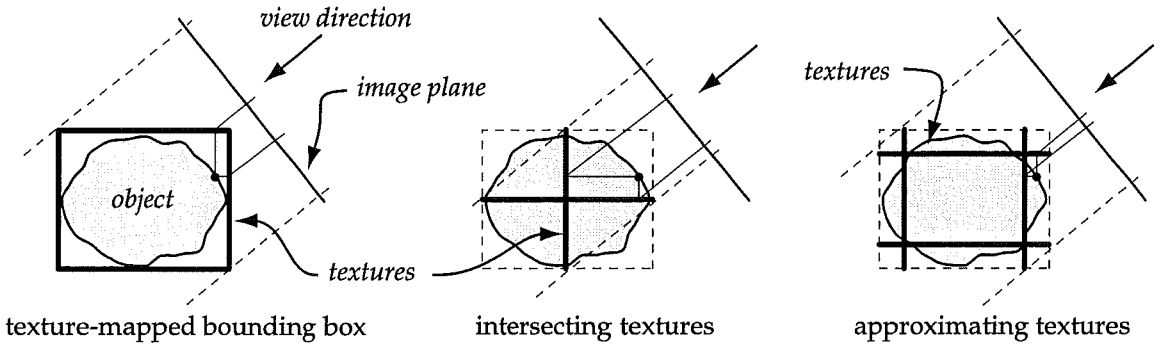


Figure 4.27: The worst-case viewpoints in 2D for the texture-mapped bounding box *(left)* and intersecting-textures *(centre)* replacements, and the modified scheme with images placed around the sides of a box replacement *(right)*.

Figure 4.28: A model of a plant *(centre)* together with two representations as intersecting textures *(left)* and with approximating textures *(right)*.

Either the m images with the smallest values of c_i, or all the images with $c_i < C$, can be chosen for rendering. This is a generalisation of the object movies used in QuickTime VR **[Chen95]** for image-based rendering of objects. Figure 4.29 shows images of a model constructed with $n = 6$, using a set of approximating, rather than intersecting, textures to give a better feeling of the object's size. Increasing n to about 14 results in smoother transitions

— a further possibility is to blend textures in and out.

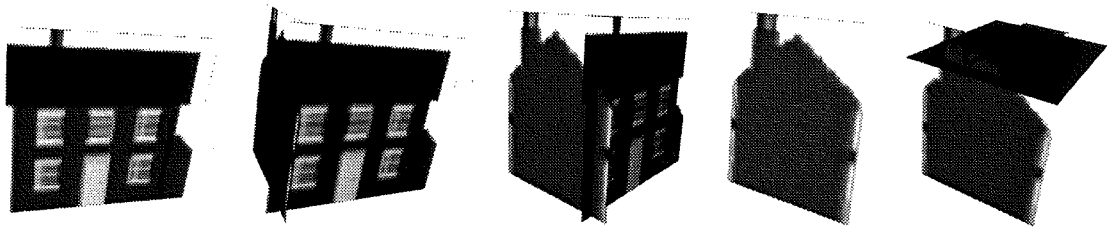


Figure 4.29: A sequence of images generated as the viewpoint moves around an object represented as a set of approximating textures from which a few are chosen to be displayed based on the current viewpoint.

An alternative use of texture-maps is as approximations to an environment [**Greene86**]. The further away the elements of an environment are from the viewpoint, the less the geometrical inter-relationship between them changes as the viewer moves around. A good example is a distant landscape or sky rendered as an image on a backdrop to a scene in the foreground [**Regan94**]. Since the display is image-based, it is independent of the complexity of the scene it replaces. Plenoptic modelling [**McMillan95**] takes this one step further and enables several such panoramas to be combined to encompass backdrops which change based upon viewer location. These background objects would only need updating occasionally — when the viewer has moved significantly — in a manner similar to the variable object update rates proposed by [**Wloka93**].

4.5 Structuring and Hierarchical Replacements

Having introduced techniques that can replace a group of objects by one simpler one, the final section of this chapter examines the last mechanisms required to complete the process of automatic hierarchical approximation. Objects need to be grouped together for input to replacement algorithms, and this section examines what criteria and algorithms can be used to structure models in an approximation hierarchy. Since grouping determines the form of approximations, the attribute values calculated in the first sections of this chapter are proposed as an original way of guiding grouping algorithms based on perception.

4.5.1 Structure in 3D models

Structuring plays many roles in computer modelling, and unless the storage mechanism allows multiple structures to co-exist over the same objects, only one of these can be chosen.

Structures already in use include:

- **modelling structure** created by the designer during modelling,
- **animation structure** controlling the relationship between moving and rigid parts,
- **visibility hierarchy** for answering visibility queries efficiently.

An approximation hierarchy can be based on any one of the above or be completely separate. Not all hierarchies will be as good as a dedicated one, since it may be difficult to find a reasonable quality approximation for a disparate group of objects. Even when using an existing structure, however, it can be beneficial to apply a grouping algorithm that extends the hierarchy, taking a large unstructured list and inserting a few levels of new structure without affecting the surrounding hierarchy (Figure 4.30).

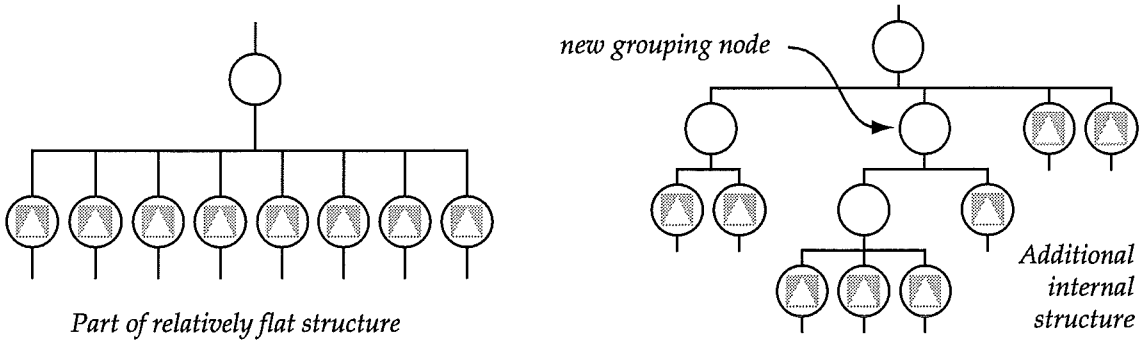


Figure 4.30: Additional "sub-structure" can be inserted into flatter hierarchies *(left)* for the purposes of approximation *(right)*.

The primary aim of a grouping method is to produce groups which will approximate well. If approximations were to be used only when objects were small enough for their form to be indiscernible, structuring would simply be a matter of optimising these groups. However, if approximations are to be used at larger scales where their form *is* visible, then grouping should aim to increase the viewer's comprehension of the objects. By choosing groupings that appear natural to the observer, a lower quality view may be sufficient.

4.5.2 Perceptual Grouping

If ten people were shown a complex scene, such as an office, and asked to group all the items in the room into clusters for simplification, there would probably be (at least) ten different responses. Some objects would be easy — books on a bookshelf for example, or a cupboard unit. Others might be more difficult — should the sheet of paper on the table be grouped with the table or with the book lying near it? What about the legs of two adjacent tables – do they belong together, or should each table be a separate group?

Theories of human perception can *suggest* how natural groups might be constructed. The Gestalt Approach [**Goldstein86**] in particular proposes many **laws of organisation** and

demonstrates that structure and pattern are important visual cues. The following are some of the laws that may be applicable in the grouping of objects for simplification, some of which are illustrated in Figure 4.31:

- **Pragnanz** (also described as "simplicity" or "good figure")
  Every stimulus pattern is seen in such a way that the resulting structure is as simple as possible.
- **Meaningfulness** Objects are more likely to be grouped in a way which is familiar.
- **Proximity** Objects near to each other appear to be grouped together, and this grouping occurs at a hierarchy of scales.
- **Similarity** Objects that are similar are grouped together.
- **Good Continuation** Series of objects following smooth lines appear grouped.
- **Common Fate** Objects moving in the same direction appear to be grouped.

While the laws involving human experience and context prove difficult to emulate in practice, those based on similarity are most promising. The shape of an object (usually judged by its profile), its orientation, colour and brightness, spatial location and motion are all possible contenders for similarity measurements [**Goldstein86**] [**Marr82**].

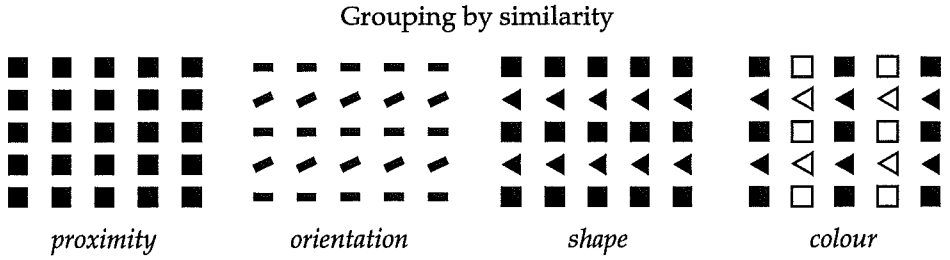


Figure 4.31: Changing the properties of shapes affects the way in which they appear to be grouped. Proximity, orientation, shape, colour are shown as examples of such influences.

Finally, human perception tends also to favour smaller groups, since in the pre-attentive stage of perception the ability to count more than three or four objects is minimal [**Chase86**]. This implies that a deeper hierarchy of small groups might be favoured.

4.5.3 Automatic Hierarchy Generation

As discussed in Chapters 2 and 3, spatial proximity should be the most fundamental criterion for grouping, since it is important that decisions about quality can be made locally. Because of the efficiencies of culling objects, spatial proximity is important in all grouping algorithms, especially for ray-tracers, whose performance is drastically reduced if a bad

visibility hierarchy is in use. Spatial proximity, however, should not be the *only* criterion used to group objects.

In order to build good hierarchies automatically, it is necessary to be able to gauge different possible hierarchies against each other in order to determine which is "better". Such judgements can be made by constructing a criterion which is evaluated over hierarchies. This will involve quickly estimating the quality of approximations in the hierarchy. Existing algorithms often base cost criteria on the expected number of intersection calculations for random rays **[MacDonald90]** **[Subramanian91]** **[Subramanian90]**. This turns out to be equivalent to finding groupings so that the surface area of the bounding volumes is minimised **[Goldsmith87]**.

Because of the huge number of possible configurations, an exhaustive search is not practical for more than about 6 objects when there are already almost 3000 groupings to evaluate (rising to over 280 million for 10 objects). A more realistic approach is to use knowledge about the objects to guide the search through the space of possible trees.

Grouping by location has been extensively studied for point data (*e.g.* **[Samet90]**). Including the extent of an object is much more difficult. Using bounding boxes instead of the original objects is one means of simplifying the problem. Alternatively, better approximations to objects can be used in the grouping method.

A two-phase algorithm based on spatial subdivision to build a good hierarchy is described in **[Glassner88]**. The objects are first placed into an octree, and then the spatial information is used to build a bounding volume hierarchy. An octree is used directly as the basis of an approximation hierarchy in **[Maciel95b]**, generating bounding boxes around the objects within each node. The problem with the spatial subdivision approach is that objects that lie on either side of a dividing line, *however close*, cannot be grouped together at that, or any lower, level. Furthermore, objects which *cross* such a boundary must be included somehow — both these approaches simply insert such objects at a higher level.

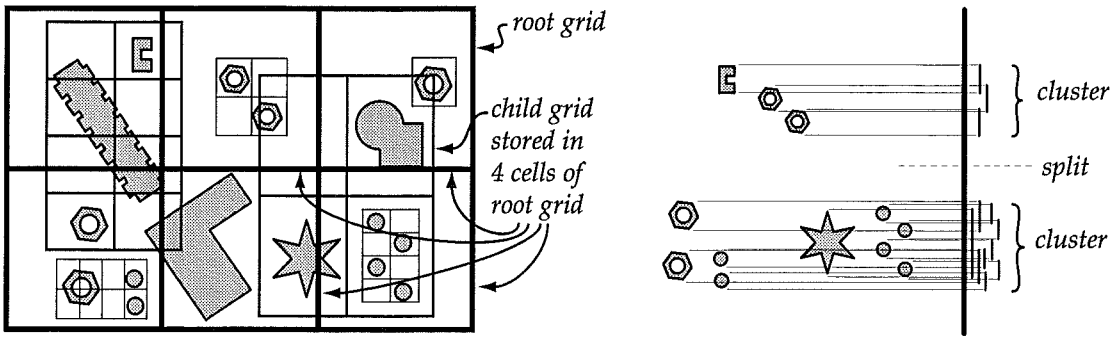


Figure 4.32: The hierarchical uniform grid (HUG) structure *(left)* and the projection method used to identify clusters *(right)*.

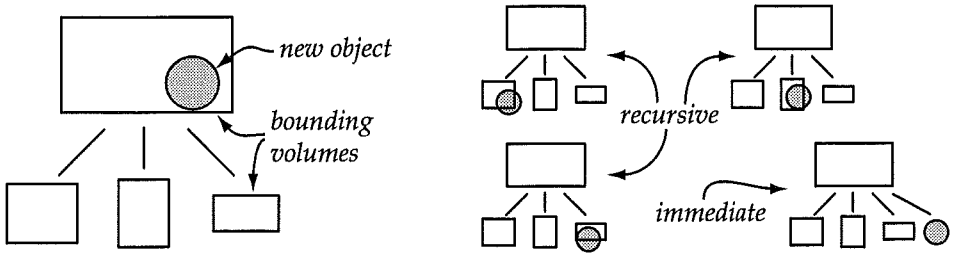
Filtering objects by a scalar quantity such as size is simple. A **hierarchy of uniform grids**

(Figure 4.32) storing objects of similar size can be built by clustering objects at each size and placing each cluster in a separate uniform grid [**Cazals95**]. Clusters are identified by examining the projections of objects on each axis in turn, splitting the set of objects where their projections do not overlap. In spite of being prone to problems with axis-alignment, the results are promising, and faster than the corresponding grouping algorithms for large sets of objects. By approximating all the objects at each level as a group, the resulting spatial subdivision structure can be used as the basis for an approximation hierarchy.

4.5.4 Heuristic-Led Object Insertion

An alternative to the bottom-up approach is a top-down one, repeatedly inserting new objects into an existing hierarchy in the best location, estimated by a heuristic [**Goldsmith87**]. The heuristic used estimates the increase in cost that would result from inserting an object into each child of the root node, and of inserting it directly as a child of the root. If one of the children is chosen, the algorithm proceeds recursively, inserting the object into that child node.

The cost heuristic, $Cost(A, B)$, calculates the increase in the surface area of the bounding volume of A when B is inserted directly into it, capturing both the spatial location *and* size of the objects concerned. While near the root node, it is likely that several branches will have zero cost, since the new object falls completely inside their bounding volumes. In this case, possible paths through the tree cannot be distinguished; all are searched until one is computed to be better than the others, an event which rapidly occurs the further down the hierarchy the object travels (Figure 4.33).



Insertion of object into hierarchy — 4 possible insertion positions

Figure 4.33: Illustration of heuristic based object insertion

The heuristic-led algorithm is attractive since single object insertion is $O(\log n)$, giving a total time of $O(n \log n)$ for structuring n objects. The original work found that the order of insertion could make a significant difference to the efficiency of the hierarchy and suggested that trying a small number of random orders and then picking the one with lowest cost is the best strategy. The 34 trees in Figure 4.34 are grouped into a hierarchy using this method with 10 random orderings evaluated in about 3 minutes. The groupings are driven by the

axis-aligned bounding boxes that are created, and it can be seen that this does not always produce an ideal structuring.

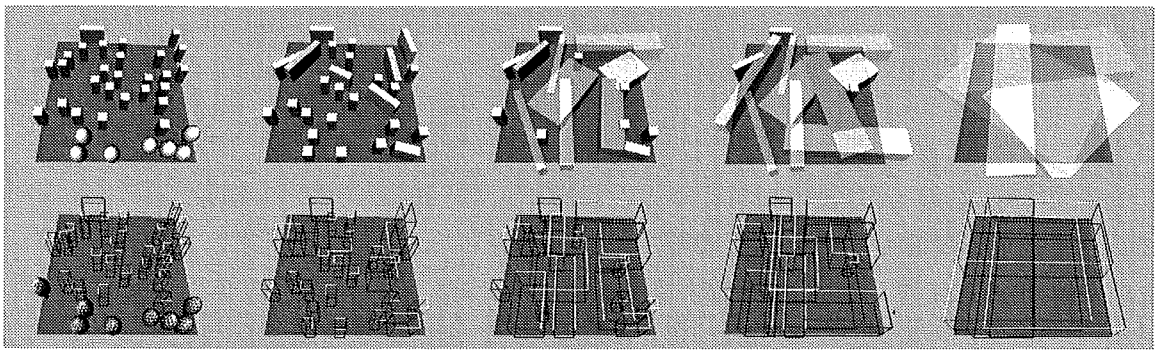


Figure 4.34: A number of randomly scattered trees *(top)* grouped together automatically by minimising bounding box sizes *(bottom)*.

The heuristic used to guide this insertion algorithm can be extended to include weightings that describe the similarity of objects in other respects. Some of the problems of using axis-aligned bounding volumes might be removed by basing the structuring on approximations. Heuristics which weight solutions that group objects of similar surface properties, orientation, or geometrical shape, calculated using the methods described earlier in this chapter, form an interesting area for future work.

4.6 Summary

The last two chapters have described many aspects of object approximation, in particular focussing on the methods required to create approximation hierarchies. The importance of perceptual-based methods for the generation of approximations has been emphasised since such considerations should lead to better replacements for use at larger scales. Two approaches to computing values for an example set of attributes have been described, and the results have been applied to calculating the geometry and surface properties of replacements as well as to choosing how to structure an approximation hierarchy. The remaining chapters examine how such data, once generated, can be used effectively.

Chapter 5

Algorithm Approximations

— In which rendering algorithms are described that take advantage of approximate data.

The previous two chapters discussed techniques for generating approximation hierarchies for models. This chapter examines the algorithms that use such approximate data, and describes how existing rendering algorithms can be altered to accommodate and take advantage of the new data.

The purpose of introducing approximation hierarchies is to enable applications to process only a subset of a complex model. The first section describes how the rendering process is modified to identify a representative subgraph (Section 2.3) of the approximation hierarchy efficiently. Sections 5.2 and 5.3 present details of how methods based upon quality criteria can be integrated into a Z-buffer based rendering system and a ray-tracer. Section 5.4 examines how approximation can be extended from the data to other aspects of algorithms, such as illumination models and radiosity methods. The final section proposes ways in which quality control can be enhanced to provide some of the benefits to the user described in Chapter 1, covering progressive and fixed-time rendering, and novel uses of quality controls as an aid to visualisation and communication.

5.1 Representative Subgraphs

Incorporating approximation nodes into an object hierarchy provides an element of choice to any algorithm using the data. This section examines how to select which representation

to use at each approximation node, thereby defining the subset that is required. Quality criteria are proposed as a means of making decisions efficiently, and the sort of additional data-structures needed internally to store approximation hierarchies are briefly introduced.

5.1.1 Subset Identification

Using a representative subgraph of an approximation hierarchy guarantees that an application has *some* representation of *all* the objects in the model. In most cases, this subgraph can be truncated by removing branches corresponding to objects that are not directly visible from the current viewpoint. Illumination models, however, often increase this required set due to calculating direct and indirect lighting contributions. Existing visibility algorithms attempt to estimate the minimal model subset required — this chapter shows how this can be extended to include detail calculations.

Most algorithms for accessing data from an approximation hierarchy $\mathcal{H}$ will require a traversal of the model, during which exactly one child is chosen at each approximation node, resulting in a subgraph $\mathcal{S}$. While a Z-buffer rendering algorithm uses only a single traversal of the hierarchy, more complex algorithms may require several traversals, defining many subgraphs $\mathcal{S}_i$. A ray-tracer, for example, will use several for each pixel in the image. For each of these traversals, a different choice may be made at each approximation node. The subgraph $\mathcal{S}$ required for the whole image now becomes the union of all the subgraphs $\mathcal{S}_i$, and may include several representations at each approximation node. This chapter aims to demonstrate that $\mathcal{S}$ is a very limited subset of a complex model due to the inclusion of detail constraints in addition to visibility restrictions, and furthermore that its size depends on the quality of the results.

At each approximation node it is necessary to make the decision between different approximations as quickly as possible to avoid a drop in performance; certainly in much less time that it would take to draw the approximation. While it is important to use the most appropriate approximation under the circumstances, it may be sufficient to make a quick choice based only on an estimate of which would be best.

Decisions are based upon three classes of **quality variables**:

- **data-dependent values** defining the quality of the data in the model,
- **context-dependent values** indicating how the data is transformed during use,
- **control values** set by the user to determine how the application functions.

For example, the effects of perspective scaling are described by context-dependent quality variables which modify the data-dependent values describing the quality of each object, and the results are compared to the control values specified by the user to determine the quality of the image produced. The combination of these values results in a **quality criterion** which is used as a means of specifying the representative subgraph required.

5.1.2 Quality Criteria

The most efficient form of quality criterion is a single real-valued variable $C(A, S, T)$ which is a function of the quality variables associated with each approximation A, the current viewing context described by the state S and the control settings T (Figure 5.1). This is defined so that the lowest quality representation that is acceptable in the circumstances has the smallest positive value c_i

$$c_i = C(A_i, S, T) \quad \text{for} \quad i = 1 \ldots n \tag{5.1}$$

corresponding to approximation A_i.

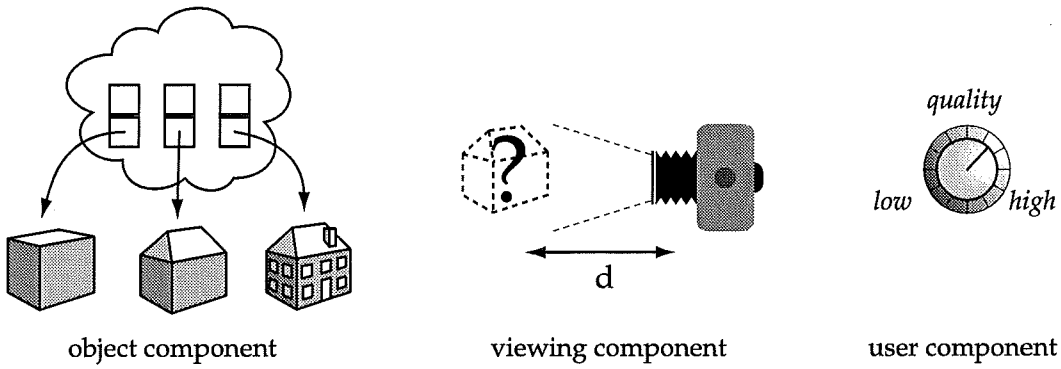


Figure 5.1: Classes of quality variables used to generate quality criteria for different approximations.

All quality control data should be pre-processed as much as possible to minimise delays at later stages. The most convenient form for the quality criterion is as

$$C(A, S, T) = T - F_s(S)\, F_a(A) \tag{5.2}$$

in terms of a factor associated with the context in which the object is being used $F_s(S)$ and a factor $F_a(A)$ depending on object quality which would have to be modified to $F_a(A, S)$ if object quality were context-dependent (*e.g.* **[Maciel95a]**). If Equation 5.2 can be applied, then the approximations A_i can be sorted in advance on $F_a(A_i)$ — the resulting algorithm is outlined in Figure 5.2. The criteria does not have to be computed for all approximations, since it terminates when the first sufficiently detailed one is identified.

The criterion need never be evaluated for the last approximation since if none of the others are accurate enough it will be used anyway. The quality values are in fact used to discriminate *between* adjacent approximations and it would be sufficient to store quality variables that separate pairs of approximations. The values of such variables, however, belong to each individual representation, and the two should remain associated, especially if the set of available approximations is changing.

The quality criteria for selecting the subgraph, F_s and F_a, depend on the application — criteria for Z-buffer based rendering and ray-tracing are described in Sections 5.2 and 5.3.

```
chooseApprox( threshold, state, approx[] )
  {
     cs = evaluateStateCriterion ( state ) ;
     i = 0 ;
     REPEAT
       {
         i = i + 1 ;
         ca = evaluateApproxCriterion ( approx[i] ) ;
         criterion = threshold - cs * ca ;
       }
     UNTIL ( criterion >= 0.0 OR i == n ) ;
     RETURN ( approx[i] ) ;
  }
```

Figure 5.2: Pseudo-code to choose one of n pre-sorted approximations based upon the criterion given in Equation 5.2.

5.1.3 Structural Support

In order to use approximation hierarchies, existing systems must first be modified to store approximation nodes. The data-structure for an approximation node must contain a list of objects together with the additional information required to choose between them. The previous section has shown that in most cases a single quality value associated with each representation is sufficient for this purpose. In many cases, a binary approximation node of fixed size, rather than an arbitrary length list, will suffice, since a multi-way choice can be simulated by a tree of binary nodes Figure 5.3.

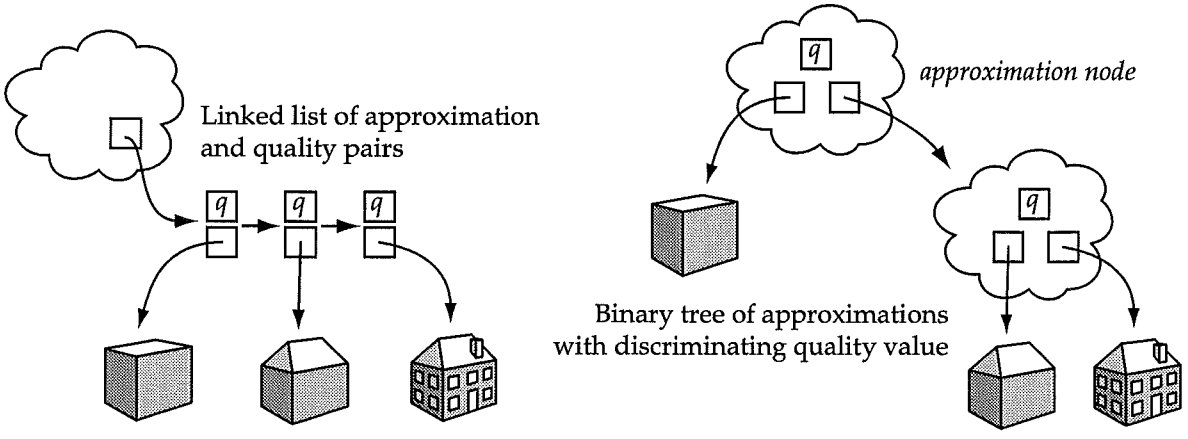


Figure 5.3: Two possible forms for storing an approximation node.

5.2 Z-Buffer Systems

This section describes the minimum set of modifications that need to be made to an existing rendering system in order to take advantage of approximate data. As an example, a Z-buffer system is chosen since, for rendering purposes, it requires a simple pass through the model. Initial implementation was in C [**Kernighan78**] on top of the low level library created for driving the Cambridge Autostereo Display [**Moore96**] [**Castle95**]. This is a 3D display device which requires between six and sixteen views of a scene to be drawn in one frame-time, albeit at a low resolution of 320×240 pixels. The system has subsequently been re-implemented in C++ [**Stroustrup91**] on top of Open Inventor [**Wernecke94**] on an SGI Indy XZ [**SGI94**] with faster rendering speed, a high-resolution display, and a much more flexible scene graph structure.

5.2.1 Subgraph Identification

The majority of Z-buffer systems compute only direct visibility and evaluate lighting models without calculating shadows or reflections so the visible subset required for generating an image contains just those objects directly visible. Each object that appears in the final image is seen directly from the viewpoint and is projected onto the image based upon the standard projection geometry shown in Figure 5.4. Its linear dimension is thus scaled inversely by the distance from the eyepoint, and this scaling is the basis for mapping object quality factors to image space.

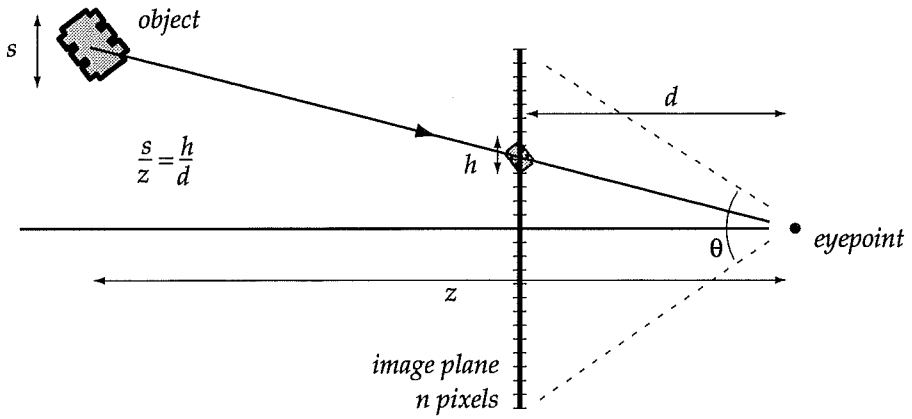


Figure 5.4: The perspective projection geometry used in calculating quality criteria.

Inverting this mapping shows that the quality of objects required decreases with distance from the eyepoint, depending only on the object's position in space. This dependency is illustrated in Figure 5.5 where the surfaces in the image on the left are coloured according to the quality at which they are accessed. The image on the right shows the same informa-

tion from a different viewpoint and demonstrates the restricted size of the required subset when limited by both visibility and detail considerations. It is analagous to the spread of **importance** through the environment described in **[Smits92]** for radiosity computations (see Section 5.4.2) but considered for Z-buffering.

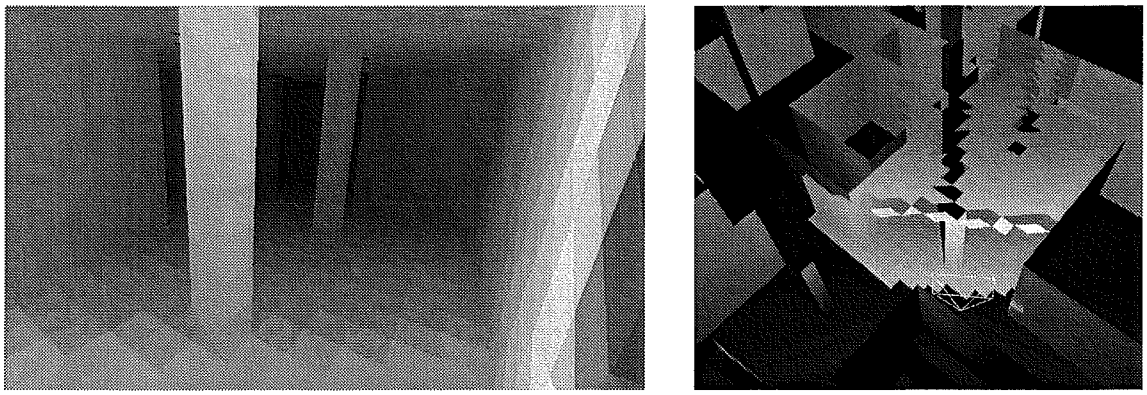


Figure 5.5: An image of a building environment *(left)* and an overview of the model *(right)* with the surfaces shaded according to the quality at which they were accessed (high-quality red, low-quality blue).

5.2.2 Example Criteria

A few rendering systems already provide level of detail nodes that use quality criteria to decide which versions of a model to draw. This section describes four types of quality criteria, each of which have slightly different computational costs and, more importantly, different interfaces to the user and model designer. After describing two common criteria, two new ones are proposed which can be expressed in the form of Equation 5.2 and provide a more useful interface for specification and control. Section 5.5.5 shows how more complex quality controls can be implemented and used to aid visualisation.

Object Screen Area

Open Inventor 2.0 **[Wernecke94]** includes a 'LevelOfDetail' node containing a sequence of children (approximations), separated by values of a 'screenArea' field, similar to the level of detail handling in RenderMan **[Upstill90]**. The bounding box of the 'LevelOfDetail' node is used to estimate an area coverage for the node from the given viewpoint. The principal way of doing this involves projecting the bounding box into screen-space, and computing the area of an axis-aligned bounding rectangle around the image of this box (Figure 5.6). A scaling may occur to the computed area depending on the setting of a global complexity value. Badly-fitting bounding boxes in object-space or bounding rectangles in screen-space

mean that approximations oscillate, causing a distracting "popping" effect as an object is rotated, since the calculation is orientation-dependent rather than object-based.

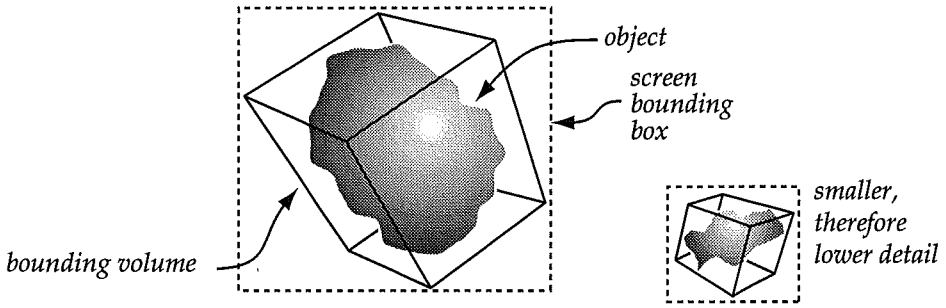


Figure 5.6: Computation of a quality criterion based on the screen area occupied by the projection of an object.

Distance to Object

VRML 1.0 and Open Inventor 2.1 include a modified 'LOD' node which uses a distance field to differentiate its children; it is similar to the 'LOD' node in IRIS Performer [**Rohlf94**]. These values are compared to the distance of the centre of the object from the viewer. This calculation has the advantage of being much faster than the one described above, and is also less prone to problems due to badly-fitting bounding boxes. A scaling based on the current complexity value is again included, although the effect of this is more difficult to understand, particularly if the field of view can be altered. Since the measure is not object-based, it is more difficult for designers to specify.

Object and Feature Importance

An alternative to computing the screen size of an object by projecting a bounding box, is to store the visible surface area (*VSA*), or **importance**, of the object and project this on to the image. This is much faster, but makes the assumption that the object is roughly the same size from all directions. Estimated screen size, s, is given (in terms of the variables shown in Figure 5.4) by

$$VSA \cdot d^2 \; / \; z^2 \tag{5.3}$$

where object size is also scaled by the square of any uniform scale factor applied to the object. Screen size can be converted into the fraction of the image occupied by multiplying by

$$f = 1/\left(4\, d^2\, \tan^2(\theta/2)\right) \tag{5.4}$$

in order to present an intuitive control value to the user which varies correctly whatever the current field of view θ. By using

$$F_s(S) = 1 / (4\, z^2\, \tan^2(\theta/2)) \quad (5.5)$$

$$F_a(A) = VSA_A \quad (5.6)$$

in Equation 5.2, the user threshold T then controls the fraction of the screen area that should be occupied by a representation before a higher quality approximation is required. The constant values in these equations can be moved into the threshold value T on a frame-by-frame basis for efficiency.

Object importance can be used quickly to decide roughly how large an object is but does not adequately encapsulate quality. The **feature importance** of an object is the maximum area of *features* on the object, and replaces VSA_A in Equation 5.6, and can be transformed in the same way as object importance to enable a rapid comparison of quality between objects.

Effective Object Resolution

A more flexible and intuitive measure than feature importance is **effective resolution** — a distance measure on objects which can be mapped onto a distance measure on the image using

$$F_s(S) = 1 / (2\, z\, \tan(\theta/2)) \quad (5.7)$$

$$F_a(A) = \text{effective resolution} \quad (5.8)$$

The user threshold T now controls the maximum linear size of the object feature on the image in a similar way to the importance value above. This is very close to the spatial frequency described in [**Reddy94**] and it defines the sampling resolution below which errors on the object are not important. The resolution measure is intuitive for model constructors since it can be defined as the maximum deviation of the approximate object from one of high quality. This criterion is used in the example systems illustrated in the rest of this dissertation.

5.2.3 Results and Performance

By using approximations at rendering time, performance improvements can be expected in a number of areas. As an example, an infinitely detailed object can be created using a recursive definition and rendered using approximation nodes which curtail traversal of the structure when the primitives are suitably small on the screen. Figure 5.7 displays two images of such a fractal at different scales showing how the primitives remain roughly the same size on the image. An attempt to draw this object accurately would never terminate and techniques required special sampling methods [**Hart91**].

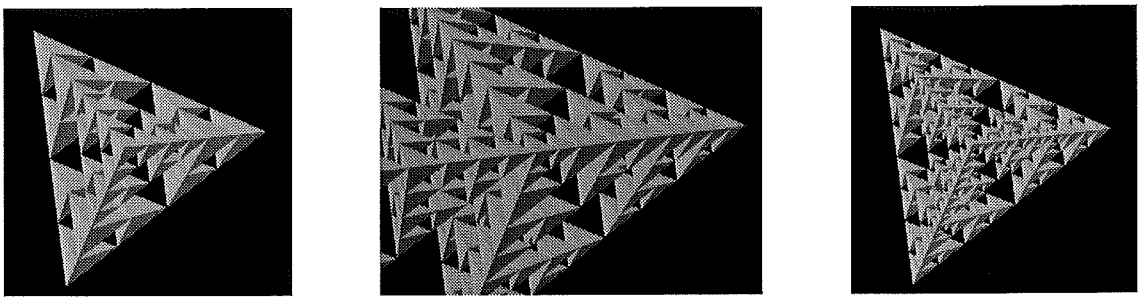
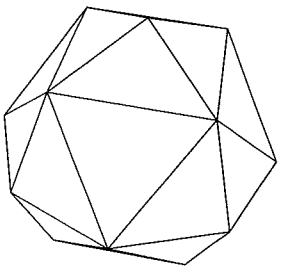


Figure 5.7: An infinitely detailed object drawn at different scales but the same accuracy *(left)* and *(centre)* and at a higher accuracy setting *(right)*.

A sphere subdivided into 32 triangles will involve roughly the same number of memory accesses as one divided many more times since it covers the same number of pixels. The only difference between the two is the number of polygons, vertices and edges processed (Figure 5.8), both representations having the same depth complexity and occupying the same screen area. Polygon count is not the only limiting feature of such systems — another important consideration is pixel accesses. Hardware rendering systems can often be limited by Z-buffer and image memory access times. In order to achieve maximum throughput, it is necessary to reduce the number of memory accesses per pixel, determined by the **depth complexity** — the number of surfaces which overlap that pixel.



	Low quality	*High quality*
Number of polygons	32	648
Average projected area	2.60	3.14
Total edge length	169	723
Total vertices	96	1944
Minimal mesh vertices	18	254

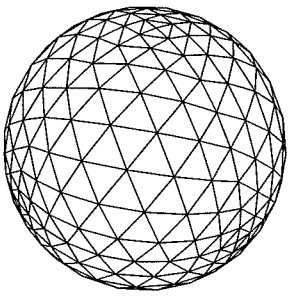


Figure 5.8: Two polygonalisations of the sphere and associated costs in terms of area and numbers of vertices, edges and polygons.

Since many objects are modelled as roughly convex surfaces, the reduction in depth complexity through different levels of detail is confined to near the silhouette edges of objects and is quite small (Figure 5.9 *(left)*). As the surfaces become rougher, the depth complexity increases dramatically (Figure 5.9 *(centre)*), and replacing a group of separated objects by a single one results in a significant drop in pixel accesses (Figure 5.9 *(right)*).

The reduction in complexity in terms of the number of triangles, depth complexity, and total screen-space edge lengths for two models (Figure 5.9 *(right)* and Figure 5.11) is given in the following table and Figure 5.10. Both show a large drop in the numbers of primitives,

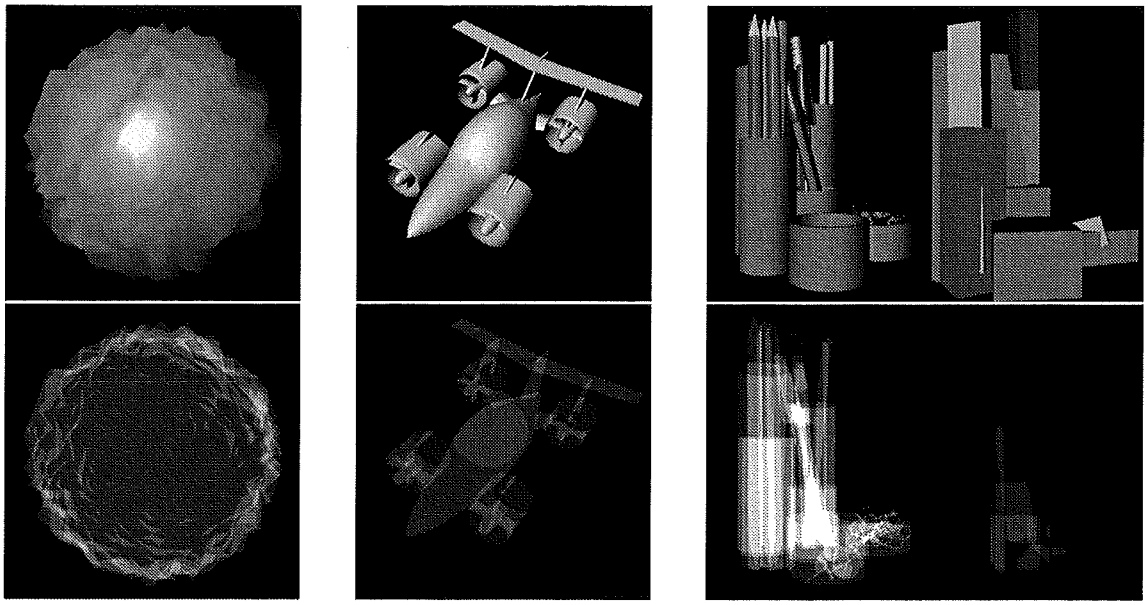


Figure 5.9: Objects together with an image showing their depth complexities on a scale from black (zero) to white (fifteen or higher) for a rough sphere *(left)*, for a typical single object *(centre)* and for a group of objects before and after replacement *(right)*.

triangles and edges. Figure 5.12 shows two images of an office (from part of a larger model) at two different qualities and a difference image indicating the location of the missing detail, which resulted in a drop of rendering time from 12 to 3 seconds.

	High quality	*Low quality*	*proportion*
Number of triangles	45608	123	0.28 %
Total edge length	3020.79	147.99	4.90 %
Total screen area	0.214	3.550	67.96 %
Average triangle area	0.000114	0.0269	234 %

Low-level support for different representations is an area which needs to be addressed. Open Inventor, for instance, attempts to create OpenGL **display lists [OpenGL93]** describing the geometrical structure of objects in a compact manner, which can then be cached by the OpenGL server. This means that the next time such an object is drawn, it is not necessary to transfer all the data again, since the server has it already. If a choice of level of detail is to be made, then the geometry *might* change each time the object is drawn, and the normal caching mode is rarely used. However, from one frame to the next there is usually only a small change in the geometry that is eventually drawn, and this needs to be communicated to the lower level software in order to take advantage of speed-ups such as caching.

As more lights are incorporated, the standard treatment of light sources in a Z-buffer system becomes difficult. These problems are common to many rendering systems and are discussed in Section 5.4.1.

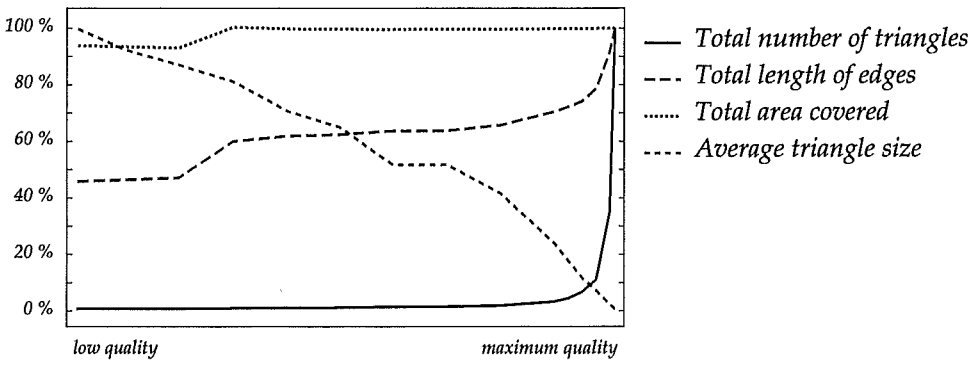


Figure 5.10: The change in rendering costs as the quality is changed (all values percentage of the maximum quality values) for the image shown in Figure 5.11.

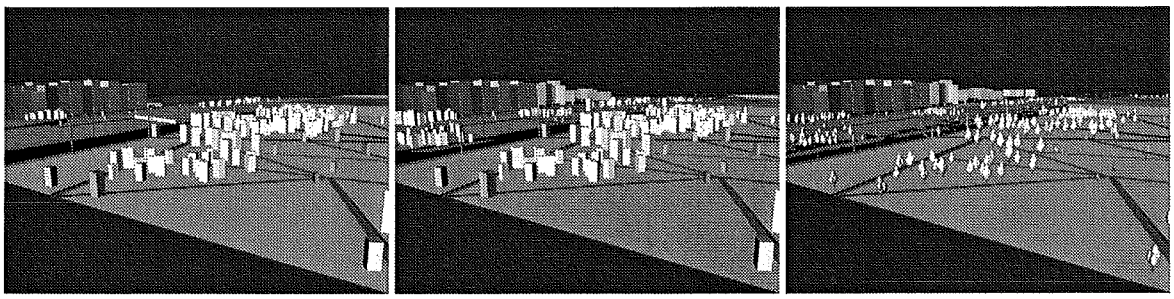


Figure 5.11: Three images of the city model corresponding to the data in Figure 5.10 with quality set to the low *(left)*, almost the maximum *(right)* and half-way between *(centre)*.

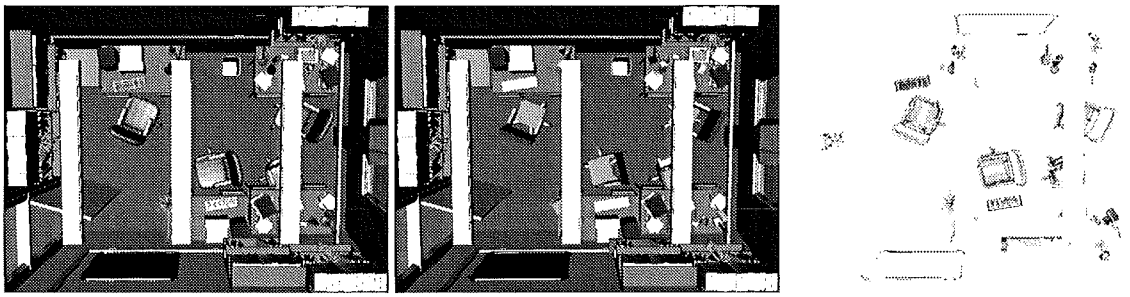


Figure 5.12: Two images of an office with 225 000 triangles *(left)* and 44 000 *(centre)* and the difference of these two images (values enhanced for emphasis) *(right)*

5.3 Ray-Tracing

Ray-tracing provides a different context for the use of approximation hierarchies. The algorithm is remarkably simple [**Glassner93**] being based around the single operation of intersecting a ray with an object, yet systems based upon it are controlled through a complicated array of quality control variables. Furthermore, ray-tracing is typically slow, being very dependent on the size of the model, and would benefit greatly from a means of producing rapid previews. By using approximation hierarchies, a uniform approach to quality can be adopted, and images of differing quality can be generated.

5.3.1 Ray–Object Intersection

A visible surface ray-tracer fires a number of rays from the eyepoint, and identifies the objects that they first intersect. The central enhancement to the basic ray-tracing algorithm is a means of intersecting a ray with an approximation hierarchy, which is very similar to the situation described in Section 5.2 for rendering approximation hierarchies with a Z-buffer.

For ray-tracing, the context-dependent part of the quality criterion depends on the ray being traced. The criteria is a function $C(A, R, T)$ of the current ray R, together with each approximation A and the quality threshold values T. Factorisation, as before, yields

$$C(A, R, T) = T - F_a(A)F_r(R) \tag{5.9}$$

with $F_a(A)$ representing the quality factor for the approximation and $F_r(R)$ that for the ray.

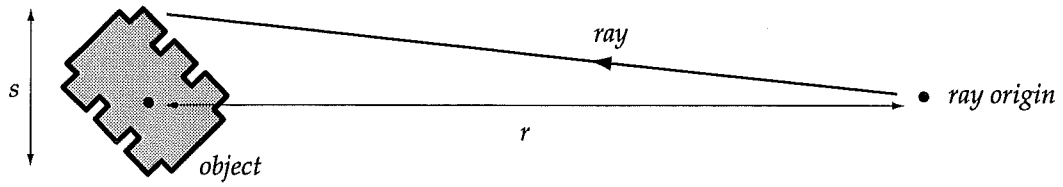


Figure 5.13: Projection geometry modified from Figure 5.4 for ray-object intersection.

Modifying the projection to that shown in Figure 5.13, yields equations for these values depending on the type of criteria involved. For example, an area-based criterion such as Equation 5.6 would use

$$F_r(R) = 1/r^2 \tag{5.10}$$

while a distance or resolution based criterion such as Equation 5.8 would be based on

$$F_r(R) = 1/r \tag{5.11}$$

where r is the distance of the object from the ray origin. Distance could be measured along the length of the ray, except this can choose different representations of objects for different rays hitting the same object. Constant terms may be included in order to scale the results to a natural value as indicated in Section 5.2.2.

These are based on a view of the ray-tracing step as an approximation to an area sampling operation. Thus for a finite area dA at distance r from origin of the ray, the area expands to be

$$dA \cdot (z/r)^2 \tag{5.12}$$

at a distance z from the origin. Inverting this, an object of size S that occurs at distance z along the ray would appear to be of size

$$S \cdot (r/z)^2 \tag{5.13}$$

at distance r (Figure 5.13). The ray quality is a measure of the contribution that the object it intersects will have on the final image, and indicates the decaying **importance** of the ray with distance, analogous to the importance introduced for radiosity systems in [**Smits92**].

5.3.2 Recursive Ray-Tracing

Recursive ray-tracing implements an illumination model that can handle specular reflection and refraction using the same basic tools as the visible surface ray-tracer. Further rays are traced from the first intersection point to evaluate incoming intensities in directions which correspond to specular reflection or refraction (Figure 5.14 *(left)*).

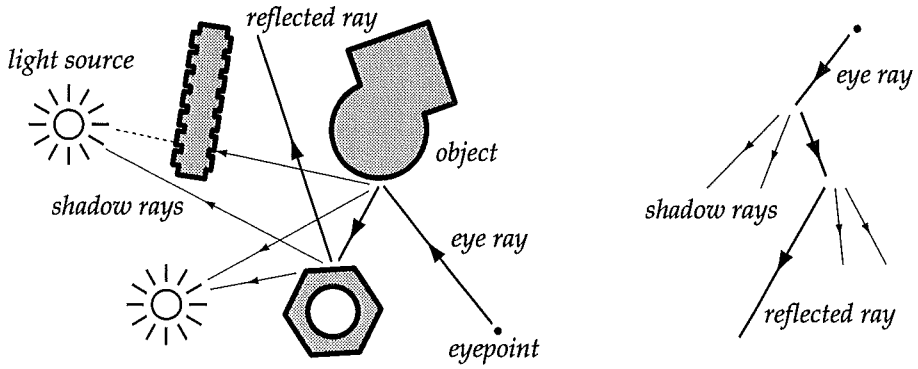


Figure 5.14: The use of recursive rays in the lighting model for each point *(left)* and the resulting ray tree *(right)*.

Since these rays are evaluated using the same method, the scheme is recursive, and the rays can be described by a **ray tree** (Figure 5.14 *(right)*). The shadow rays are used to decide whether a point is illuminated by each given light source and are discussed in Section 5.3.4.

The rays considered are essentially a sample of all the possible light paths from any light source to the eyepoint. More recent methods use even more rays in order to include phenomena such as diffuse reflection using the same basic mechanism **[Kajiya85a]** **[Ward88]** **[Wrigley93]**.

Since the basic operation remains the same, approximation hierarchies can be included immediately into a recursive ray-tracer. The only modification involves the specification of ray distance which is now defined as the total path length from the eyepoint, since the recursive rays are essentially extensions of the original ray. Figure 5.15 illustrates how a finite area expands due to multiple planar reflections. Thus when tracing a ray that follows a series of n segments of length r_i the ray length is

$$r = r' + \sum_{i}^{n} r_i \tag{5.14}$$

where r' is the length of the segment currently being processed.

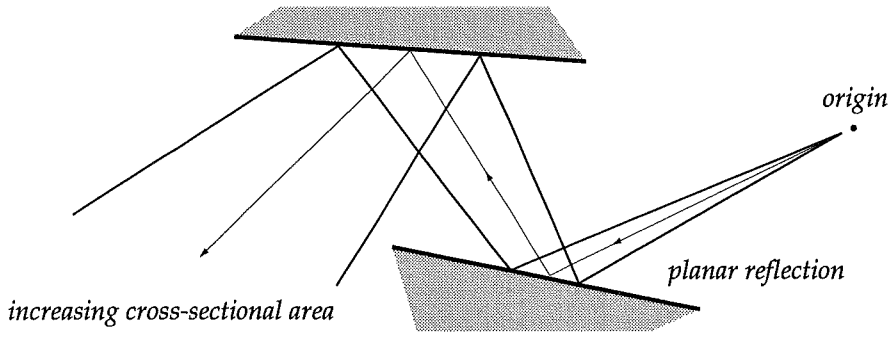


Figure 5.15: Area expansion of a ray through planar reflections.

This justification fails for convex or concave reflections and also for refractions (Figure 5.16). The quality measure attempts to capture the size that objects would appear on the screen, possibly via several reflections, and if these reflections are non-planar then the object can be scaled by more (or less) than the normal perspective scaling. For reflection from a convex surface this will simply mean that higher quality intersections are returned than might be necessary, but for concave surfaces the opposite is possible.

It is feasible to recalculate r to account for these effects using the local surface curvature to estimate a new virtual ray origin along the image of the ray. Note that by including the possibility of focussing, then an object exactly at the point of focus is accessed at infinite detail, and needs to be dealt with carefully.

Not all the rays considered in a recursive evaluation of a lighting model contribute *equally* to the resulting pixel colour. This is because the intensity carried by each ray is weighted by one (or more) coefficients, modelling, for instance, the reflective properties of the surface which has reflected the ray. This reduces the importance of the ray further — if the intensity

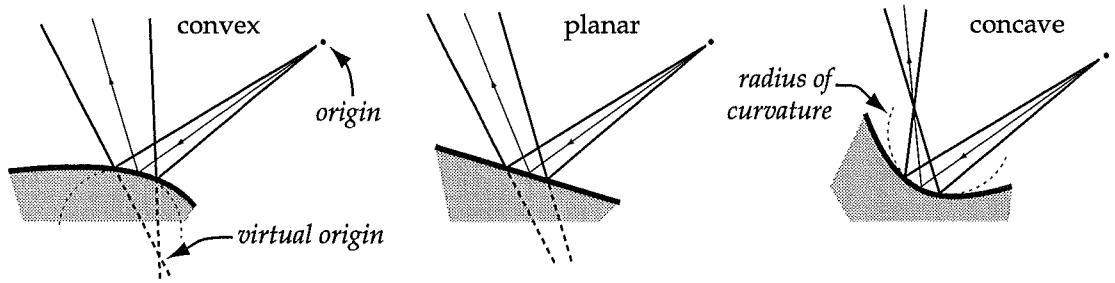


Figure 5.16: Details of reflection geometry from convex and concave surfaces, and an indication of a virtual ray origin for the new ray segment.

resulting from evaluating a ray subtree is scaled by a coefficient $k \in [0, 1]$, then the ray quality is decreased by the same factor so that Equation 5.10 becomes

$$F_r(R) = k/r^2 \tag{5.15}$$

5.3.3 Adaptive Depth Cutoff

Potentially, the ray tree is infinitely deep since, apart from shadow rays, it can only terminate at surfaces with a zero coefficient of reflection. It is usually limited artificially, by either restricting the depth to be less than a constant, or using **adaptive cutoff** to prune branches whose total contribution is below a certain value, or fraction of the total. This approach can introduce sampling bias **[Kirk91]** since energy is always underestimated, but it can be ameliorated by sampling lower quality approximations.

Apart from local effects due to concave reflections (or convex refractions), as a ray progresses through the scene, hitting a number of surfaces, it meets lower and lower quality objects. This is because the ray quality diminishes along its length, and is further reduced due to scaling by reflection coefficients and it is used to naturally prune the ray tree.

A situation which may arise after reflection is that the ray origin is located *inside* an approximation (Figure 5.17). This happens when there may be a sudden drop in ray quality due to a low reflection coefficient so that the reflected ray is intersected with a much lower quality representation than the original ray hit.

There are several possible ways of proceeding:

- Ignore all such objects for intersection calculation, which is equivalent to assuming that the accurate reflected ray misses all objects represented by the approximations ignored.
- Treat this as an intersection, but terminate the ray tree at this point since, without a surface intersection, lighting and reflection models make no sense.

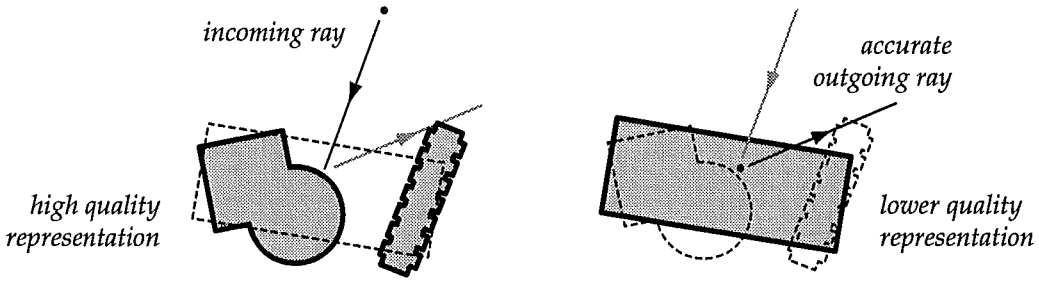


Figure 5.17: A reflected ray has a lower importance than its predecessor and its origin may be inside the suitable approximation.

- Override the choice criterion and intersect with a more accurate model of objects that have this property. Such a model need not be more accurate than that used for the previous intersection.

While the last of these is the most exact, the results of following one of the first two would be acceptable in most cases. An evaluation of the current ray quality value might also lead to a means of choosing when to apply which approach.

5.3.4 Shadow Rays

The final major class of ray intersection calculations that need to be performed are for shadow rays whose origin lies at $\boldsymbol{l}$ on a light source L and extend to the intersection point $\boldsymbol{p}$. The contribution that the light source would make (if visible) to the final pixel value can be calculated and used to control the weighting of the ray. This will be of the form

$$k \cdot a \cdot \text{ light intensity} \tag{5.16}$$

where the factor $k \in [0,1]$ is due to reflection coefficients, and the factor $a \in [0,1]$ is due to the attenuation of the light source over the distance from $\boldsymbol{l}$ to $\boldsymbol{p}$ (Figure 5.18). This ray weighting is in addition to the current quality factor for the ray which spawns the shadow ray.

The ray should then be treated as if it is an eye ray from $\boldsymbol{l}$, from which the distance r is measured, and the quality value altered from Equation 5.10 to

$$C_l(L) = k \cdot a \cdot \text{ light intensity } / \, r^2 \tag{5.17}$$

The effect of this is to deal accurately with objects that are near the light source since they will have a large effect on the location of shadow boundaries at a distance, while the objects near the intersection point can be dealt with less precisely since their effect on shadow location will be smaller.

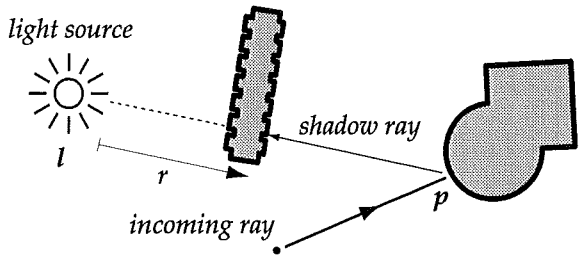


Figure 5.18: Shadow rays test the visibility of each light source from a particular point in the scene.

5.3.5 Results

Figure 5.19 is an image of the same environment as Figure 5.5, page 116, but with the rays used in a ray-tracing shown decaying in importance away from the eyepoint and after multiple reflections. Objects in the environment may now be accessed by different rays at different quality levels, but it can be seen, however, that the majority of rays of significant importance remain within a local area. This means that the required subset, although larger than that for direct visibility, remains limited in extent because the importance of objects decreases rapidly with ray length.

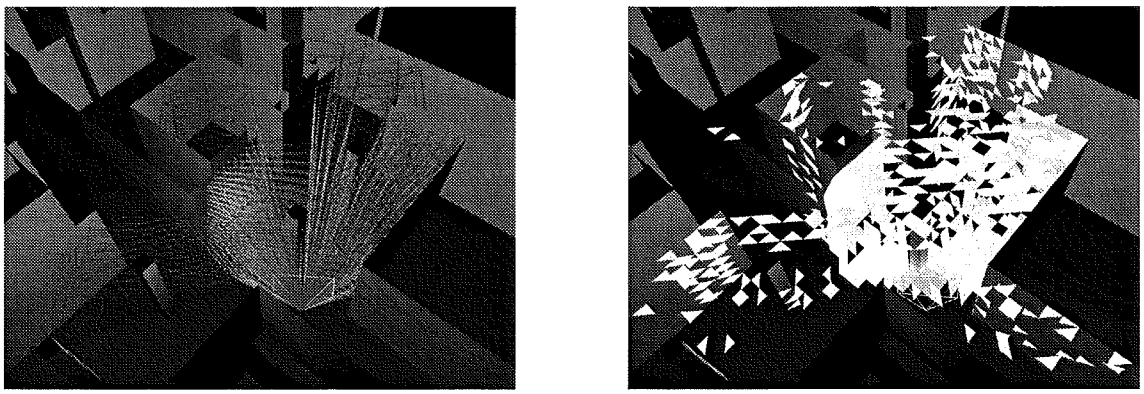


Figure 5.19: The building environment showing the rays traced *(left)* and with surfaces shaded according to the maximum quality at which they are accessed *(right)*.

Figure 5.20 demonstrates the use of lower quality representations for reflected objects and shadows in an artificial test scene, while Figure 5.21 shows a sequence of images of increasing quality of a recursively defined fractal snowflake. The two light sources are of different intensities, and cast shadows of different representations of the snowflake. The quality of the reflections of the parts of the object in itself can also be seen to increase. Rendering time multiplied by a factor of about 7 between each image in the sequence.

Figure 5.20: A simple scene where the sphere has alternate representations as a polygonal approximation and a cube *(left)*. The reflected image and the shadows use these lower quality representations *(centre)* compared to the full accuracy result *(right)*.

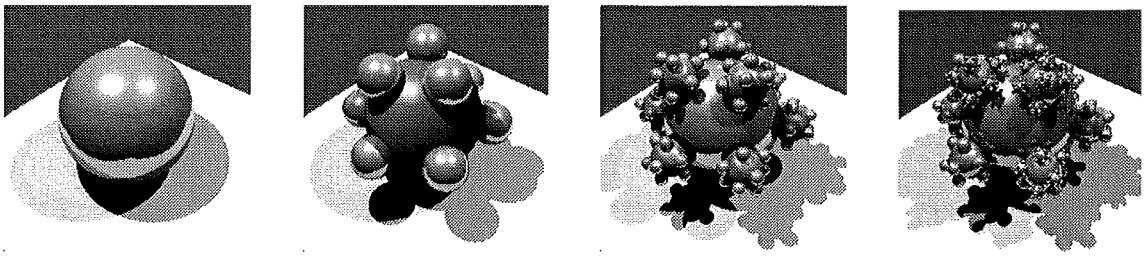


Figure 5.21: A series of ray-tracings of an infinitely detailed object at increasing quality thresholds.

The use of **amalgams** — bounding-volume replacements with semi-transparency to reduce the over-estimation of size — has been shown, in a simplified cost analysis, to reduce the costs of intersecting rays from logarithmic in the number of primitives in the model to logarithmic in the number of approximations considered [**Thompson91**]. This result is also expected using the more complete approximation framework described here, since each ray fired sees only an approximate scene, and the level of approximation is controlled by the initial ray quality. By ensuring that reflected and shadow rays have initial qualities consistent with the significance of their results, the costs have been minimised. The area in which most unnecessary computation still persists is the treatment of light sources, which must *each* be considered individually for every ray intersection point.

5.4 Approximating Algorithms

Aside from the model data used by the system, there are many other areas where a consideration of the quality of the results can lead to either less work or more appropriate computation. These include other data that may be generated by the application during execution, the accuracy with which algorithms perform calculations, and also the choice of algorithms used in the first place. The challenge is to coordinate all these choices so that there is a uni-

form approach to quality.

There are uses of model data other than rendering that can benefit from approximations. Collision detection is a good example since it is expensive to perform for complex objects. Generating approximations is another case where, as mentioned in the previous chapter, using existing approximations is often adequate. In fact, *all* users of model data should be able to function with approximate data.

5.4.1 Lighting Models

Lighting models calculate the light distribution falling at a point in terms of the light transported from a number of **light sources**. Most light sources, such as point lights or Warn's spotlights [**Warn83**], are simplifications of those found in the real world, although attempts have been made to model realistic sources like skylight [**Nishita86**], or artificial sources such as theatre lighting [**Dorsey91**]. The transport of light is also usually simplified. Most Z-buffering systems, for instance, deal only with direct light paths (without reflection *etc.*) and ignore shadows caused by objects blocking the path of the light. Radiosity methods (Section 5.4.2) attempt to account for incoming diffuse radiation from area sources by modelling diffuse inter-reflection, but at great computational expense.

As the size, especially the physical spread, of a model grows, it can be expected that the number of light sources will also steadily increase. Most rendering systems deal individually with all light sources. For instance, a ray-tracer decides whether *each* light source is visible (or what proportion of the source is visible for an area light source) at every ray-object intersection, while a polygon renderer includes the contribution of *every* light source at each polygon vertex. As the number of sources increases the performance of such systems decreases.

In the real world, as additional sources are considered, the lighting contribution for a point often does not change significantly. This is due to the intensity dropping off with distance, and the higher chance of sources being obscured (Figure 5.22). For a system that ignores shadows, however, the intensity of images will tend to increase as more sources are added, since no blocking occurs. For example, extending a model of an office to include the whole building of 100 offices will roughly increase the number of light sources by a factor of 100 but, since the lights will be mainly isolated inside rooms, the illumination in the original office *should* change very little. Unfortunately this will not be the case unless a method is used to compute the shadows of objects blocking the spread of light.

Controlling the lighting quality by *ignoring* dim or distant light sources suffers the same problems as ignoring objects — the total light diminishes, possibly reducing to zero. Light sources and their effects should therefore be approximated. Pursuing similar lines to object approximation, light sources can be both simplified and replaced (Figure 5.23). Simplification includes replacing an area-source with a point-source when it is distant, whereas replacement involves taking a number of sources and replacing them with one source that

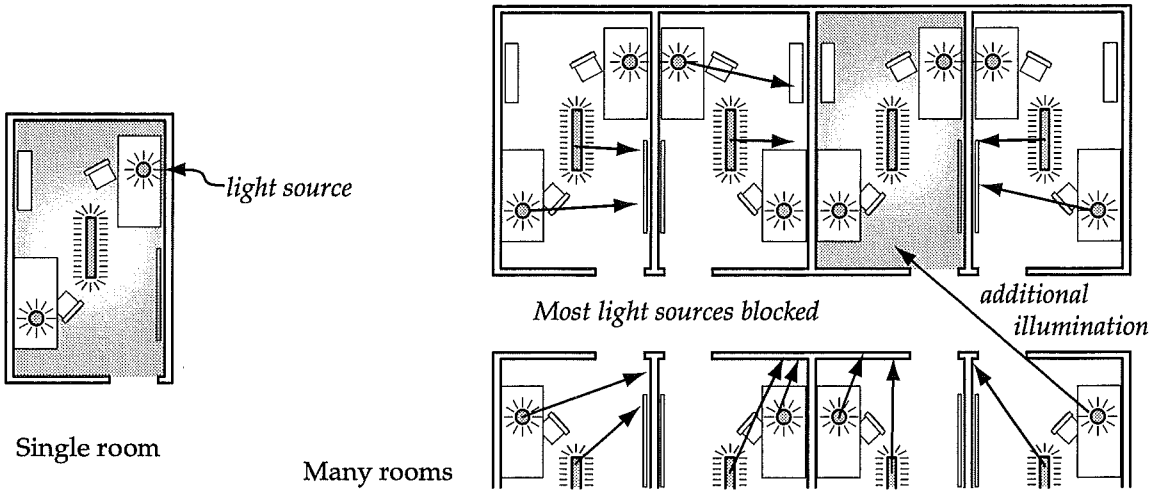


Figure 5.22: As the size of the model grows from the original *(left)* to include cover a larger area *(right)* the number of light sources increases, although the illumination in the original room barely alters.

has a similar effect.

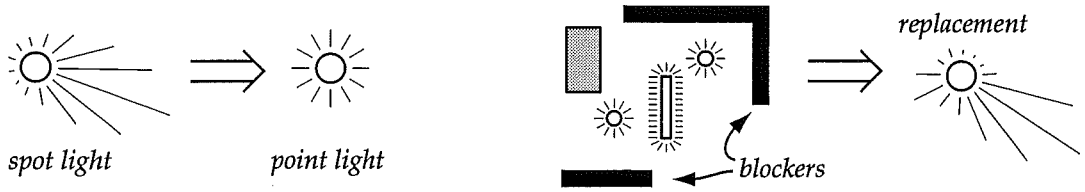


Figure 5.23: Approximating light sources by simplification *(left)* or replacement *(right)*.

Replacing a group of lights by a single source causes a local change in position, and any objects very close to the original sources, such as light shades, will cast very different (and probably unwanted) shadows. One remedy for this is to take the local blocking objects into account when replacing the light sources. This could be done, for example, by approximating the intensity distribution of the group of sources at a distance by a point-source with intensity $I(\theta, \phi)$, depending on the angle from which the source is viewed. Wavelets on the sphere **[Schröder95]** are possible ways of storing illumination distributions as multi-resolution versions of shadow depth maps **[Reeves87]** **[Segal92]**, or by approximating shadow BSP trees **[Chin89]**. Such a technique has the advantage of performing some of the shadowing functionality that is missing from Z-buffer systems, and which causes problems in complex models. The approximation of light sources is an area which needs further work, and is closely linked to the latest developments in radiosity research.

5.4.2 Radiosity Systems

Radiosity systems attempt to model the interchange of light energy between all surfaces using a purely diffuse reflection model **[Goral84]** **[Cohen94]**. Extensions have be proposed for non-diffuse situations, usually with a significant increase in computational load **[Sillion89]** **[Sillion91]** **[Wallace87]**. A radiosity solution is run in advance of any rendering, and solves for light distributions over *all* surfaces in the scene. These are then displayed with or without further lighting calculations.

Each surface is subdivided into many **elements** in order to capture the changes of intensity (Figure 5.24). Using a hierarchical decomposition enables the design of an algorithm that takes $O(\log m)$ time **[Hanrahan91]** **[Aupperle93]** for each surface where m is the number of elements, rather than the previous $O(m^2)$ algorithms. This **hierarchical radiosity** algorithm is still $O(n^2)$ in the number of input surfaces, however, and for complex environments with large numbers of surfaces this is prohibitively expensive.

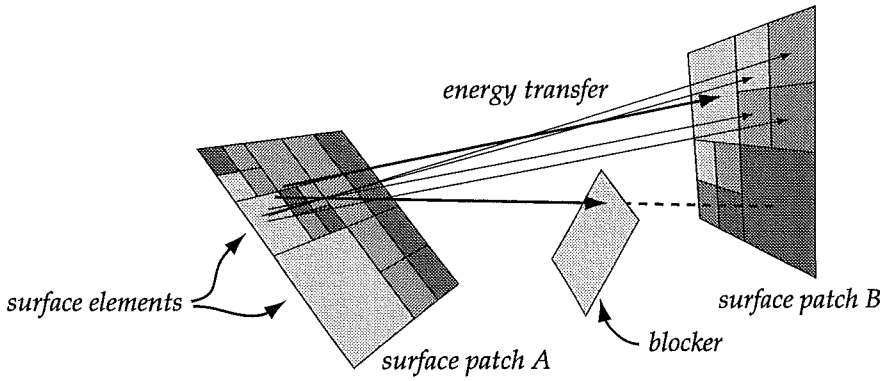


Figure 5.24: The basic geometry and interactions considered in a radiosity system.

For large and complex models, the interaction between most pairs of surfaces tends to become insignificant when considered individually, although the total contribution of a number of patches is still important. One common approach to reducing complexity is **partitioning**, artificially separating the model into cells by inserting fake surfaces between them **[Teller94]**. The radiosity method is iterated within each cell individually, and then the light incident on the boundary surfaces is re-distributed across to the neighbouring cells (Figure 5.25). This works well for distant surfaces, but patches near the boundaries end up with contributions from each other even when mutually invisible.

Attempting to accurately determine the light distribution for an *entire* model is no longer a viable approach for a complex model. Accuracy and locality needs instead to be constrained by the viewing situation. Most radiosity solvers allow control over the accuracy of the results, often by specifying a termination criterion for the iterative algorithms. **Importance-driven** radiosity techniques **[Smits92]** **[Pattanaik93]** vary the level of the solu-

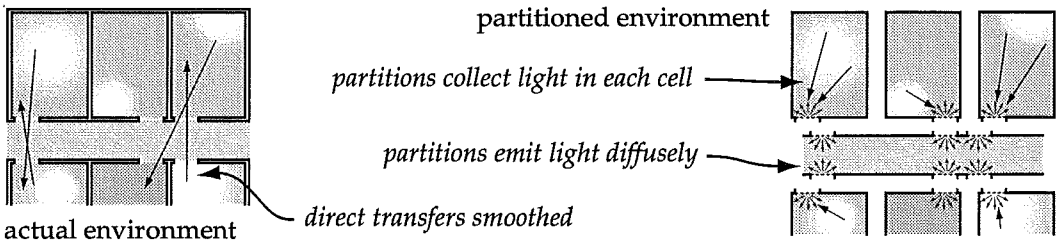


Figure 5.25: An illustration of the problems with radiosity solutions in artificially partitioned scenes.

tion over the model so that a rough solution can be used at a distance from the viewpoint. Importance is distributed around the environment from the eyepoint carried along light paths as described in Section 5.3 except via diffuse inter-reflection. Although this limits the mobility of the user, it does localise and reduce the computation, and in combination with the ability to refine areas of interest as the user approaches, might yield an efficient approach.

Two recent systems attempt to reduce the $O(n^2)$ dependency, using **clustering [Smits94]** or **volume extinction [Sillion95]**. Clustering approximates the light energy emitted from a group of surfaces by replacing the individual contribution of the surfaces in two ways. Volume extinction uses approximations consisting of volumes with a computed density to represent an isotropic collection of small surfaces.

Radiosity solvers provide plenty of scope for the inclusion of approximations and low-resolution environments have been suggested as appropriate for indirect lighting calculations **[Chiu94]** **[Rushmeier93]**. At a low level, the use of approximations can reduce the time taken in visibility calculations by matching the level of approximation to the accuracy of the radiosity computation at that stage. Both Z-buffer and ray-tracing techniques are used to solve the visibility issues for radiosity, and the techniques of Section 5.2 and Section 5.3 apply.

The approximation of light emitters is a complex topic and is discussed in the previous section. The recent introduction of wavelet techniques **[Gortler93]** for storing and accessing radiosity values, formalises the notions of hierarchical approximations within surfaces. Creating a similar hierarchy across *groups* of surfaces, and the use of high level approximations to such groups, will be very important if radiosity solutions for complex models are to become viable.

5.5 Quality Control

The previous sections have indicated how quality can be controlled *within* a system. This last section proposes ways in which additional benefits can arise by more powerful means of *user* control. The **progressive refinement** paradigm encourages the gradual and interruptible improvement of the quality of the results. Alternatively, **fixed-time** rendering lets the user specify the time in which the next image is to be generated, and leave the system to set a quality level that can achieve this. Although difficult problems, the structure of approximation hierarchies are shown to provide good starting points for both of these. Finally, a variety of quality modifications and applications are proposed that aim to access the communication potential offered by complex models and computer graphics.

5.5.1 Progressive Refinement

Radiosity systems are often constructed in such a way that intermediate results can be viewed throughout their operation, allowing the user to see low-quality images whilst waiting for high-quality ones [**Cohen88**] [**Chen91**] [**Puech90**]. This is a good example of **progressive refinement** — *i.e.* a system that produces results of increasing quality as time progresses. The main benefit derives from such a system being *interruptible* – if the user can halt this refinement process at any time then the speed/quality trade-off point does not need to be decided in advance, but can be chosen on the basis of the partial results on display.

An early example of such a system [**Bergman86**] produces images in four stages, first drawing a point cloud, then a wireframe, then including faces, first with flat shading and finally with smooth shading. The results of the previous stage remain visible, but are slowly overwritten. Another example is in an architectural walk-through application, where images are refined when the user is stationary, but when in motion, only low quality images are drawn.

Progressive refinement systems bear a resemblance to the manner in which humans perceive the world. It is natural to see broad features "at first glance" and then to concentrate for a longer time looking for detail. The brain has only limited power to interpret its surroundings, and adapts to this by adjusting its scope from the wide picture to the small feature. An artist often "progressively refines" a drawing or painting, first concentrating on its broad forms, and only later adding the details.

5.5.2 Approximation Hierarchies for Progressive Refinement

Implementing progressive refinement systems is difficult. Problems arise creating a series of results whose quality increases in a reasonably even manner *without* requiring significant extra time to generate the intermediate results. Furthermore, it is also important that

quality should increase *evenly* — a ray-tracer producing a high-quality image one pixel at a time from the top left of the image would not qualify as "refinement".

Approximation hierarchies provide a solution to the first of these problems by providing a means of varying the quality of the image in a controllable manner, allowing images to be redrawn at different qualities. Ray-tracing is one of the few rendering methods that are directly amenable to progressive refinement. Tracing pixels in a quadtree ordering produces a progressive result (Figure 5.26) rather similar to **beam tracing [Heckbert84] [Hanrahan86]**. Furthermore, the early rays which will represent a large "pixel" can be traced through a much lower quality representation of the model, and the additional cost will be small compared to the total. This extends the adaptive progressive refinement process which is only usually used for pixel-level anti-aliasing **[Painter89]**.

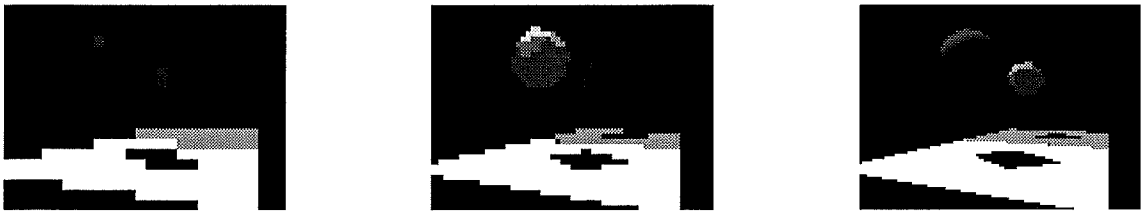


Figure 5.26: Three ray-traced images as the pixel subdivision increases and quality threshold decreases (low-quality approximations false-coloured).

Ray-tracing can also be combined with faster rendering methods by improving a lower-quality image one pixel at a time. Using a pseudo-random pixel-coverage slowly introduces shadows and reflections (Figure 5.27). This leads towards the frameless rendering approach advocated in **[Bishop94]** where pixels are generated in real-time during viewer motion, and the variable update rates for different objects advocated in **[Wloka93]**.

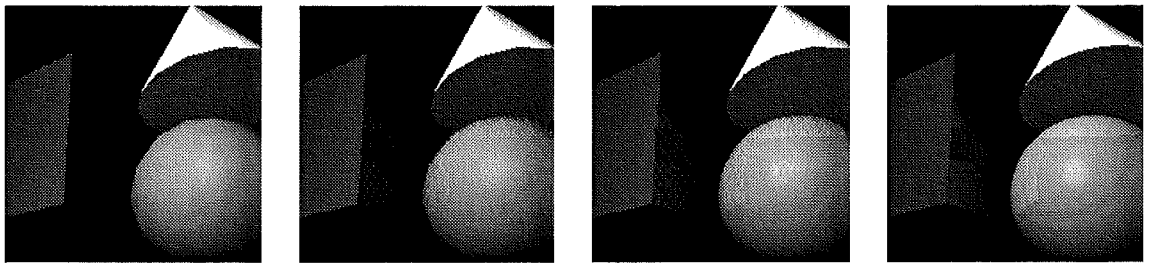


Figure 5.27: Four stages in the transformation of a Z-buffered image *(left)* to a ray-traced one *(right)* by sampling the image in a pseudo-random order.

5.5.3 Fixed-Time Rendering

Generating images in a specified amount of time is especially important for interactive systems, where frame update rates of about 10 frames per second are necessary in order to provide interactivity [**Funkhouser93**]. One of the most important aspects is to maintain a roughly *constant* frame-rate which gives users the best chance of adapting to the system.

For non-interactive applications, it also useful to be able to control the system in terms of the time it takes to produce the image. For instance, a designer might want to be sure of having the image complete in ten minutes or an hour, without having to guess which parameter settings are required.

There are two main approaches to generating images in a fixed time. Both involve estimating how long it will take to render the next image in advance, but differ in what data is used to make this estimate.

- **Reactive methods** use the rendering time for the previous frames, $r_i, \ldots r_0$ to obtain an estimate for the time to render the next frame, r_{i+1}. The values of control variables in the previous and current frames can also be used in this estimation.
- **Predictive methods** attempt to predict the time for the next frame by considering the tasks involved and estimating the time that each of these will take.

The reactive method has a number of disadvantages:

- Sudden changes in the number of objects that are visible make the time for the previous frames a poor estimate for the time for the next frame.
- The adjustment stage is very prone to overshoot and oscillation so it takes several renderings to converge. Unfortunately, if the viewpoint is continually changing, then such settling never has a chance to occur.
- Control is only available at a coarse level – it is difficult to change quality on a per-object basis.
- Lastly, this method is not applicable to single-frame images, since there are no previous frames on which to base the estimation, and similarly the first frames in a sequence need to be treated separately.

Predictive methods have their own disadvantages; the main one is that the method has to be tailored to the platform on which it is running in order to make the predictions. Furthermore, it may be difficult or even impossible to formulate costs for a complex situation in a suitably simple manner, particularly if there are multiple processors, or real-time input, involved.

5.5.4 Approximation Hierarchies and Frame-Rate Control

A **predictive method** has been proposed in [**Funkhouser93**] to pick a subset of objects for rendering within a given time. The choice is made from sets of **tuples** (O, L, R) representing a particular object O at level of detail L and rendering method R. Each tuple is assigned a **benefit** and a **cost** computed via heuristics, and a constrained optimisation is performed to maximise the benefit whilst keeping the total cost under a limit. The final algorithm is an incremental one which repeatedly inserts and removes tuples from the list used in the previous frame, keeping the conditions satisfied. The coherence between frames ensures a rapid convergence time. Variations on this approach for a hierarchy have been proposed in [**Maciel95a**].

Extending this to an approximation hierarchy, rather than a flat list of tuples, offers an immediate benefit in that no complex searching has to be performed in order to check that exactly one representation of each object is chosen. The subset used is actually just a representative subgraph from the approximation hierarchy which only changes by the substitution of one representation of an object for another. This also removes the restriction present in the original method that prevented objects being removed once inserted, and which limited the flexibility to just adjusting their level of detail.

The quality control framework advocated here provides an ideal basis for a **reactive method**, since the full range of the quality parameters should enable a complex model stored in an approximation hierarchy to be varied from full detail to just a single object. Rendering time is an unknown function $T(V, \boldsymbol{q})$ depending for a particular model on the viewpoint V and the settings of quality variables $\boldsymbol{q}$. A reactive method attempts to guess this function, based upon its previous values, in order to invert it and choose a value for $\boldsymbol{q}$ which will yield the required rendering time (Figure 5.28). The assumption is usually made that $T(V, \boldsymbol{q})$ will vary smoothly, particularly as the viewpoints change, and these methods often fail when this is not so, for instance when entering a room.

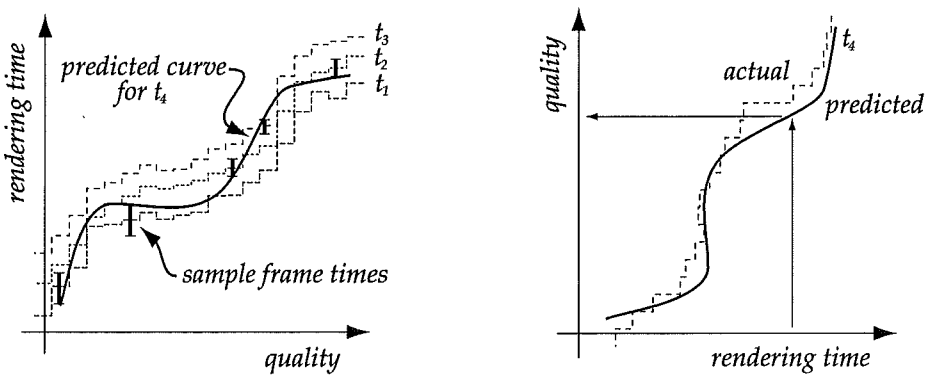


Figure 5.28: The task of a reactive method is to estimate the rendering time as a function of the quality parameters *(left)* and invert this to guess a quality setting *(right)*.

While providing the tools for controlling the trade-off between time and quality, approximations and quality control mechanisms do not yet provide a means of specifying in advance the time that an image synthesis stage should take, only the quality of the results. The methods such as those described in this section are only an early attempt to solve such a problem, but the use of a hierarchical structure is a useful aid in providing the user with more control over the application, and this is an area that would benefit from further investigation.

5.5.5 Quality Modifications

Quality control does not have to remain uniform, but can be varied in response to other factors. These effects involve modifying either the object quality or quality control variables at a finer granularity than the whole image.

Moving Objects

Another area of research concerns degrading the quality of moving objects [**Reddy94**]. Motion can occur either from a dynamic object in the database, or due to the motion of the viewer which is of interest to VR researchers who wish to reduce latencies when the user is moving. The relative speed of each object with respect to the viewer is dependent on the distance of the object from the viewer, the direction and magnitude of the object's velocity, and the directional and rotation motion of the viewer. This can be calculated and used to modify the quality threshold used for each object during rendering.

Light Energy

Approximation criteria might also take into account the intensity of an object when deciding what detail level to use. An object in shadow, or of a dark colour, could be used at a lower quality than one that is of a light colour or in an area of intense light. In general, the expected intensity needs to be estimated by looking at the lighting situation and surface properties of the object. Unfortunately, while a dark object that is placed in front of a light surface will not require great detail in its interior, the silhouette edge should be of high quality due to the contrast with the background. In general, such considerations are difficult to encode, although a scheme using the surface normal to alter quality has been described [**Becker93**].

Object Importance

One of the opportunities that is presented when objects are individually tagged with quality values, is that these can be used to enhance the quality of certain sets of objects. For

instance, if a user has an interest in a particular set of objects, such as cars in a driving simulator, the quality values for these objects can be altered to reflect this interest, while the thresholds for other objects in the scene, such as the scenery, can be reduced.

One fast means of implementation is to associate quality changes with branches of the scene graph, so that all objects in that particular group have an increased importance. This can be achieved by temporarily modifying the quality threshold whilst traversing such a subgraph, and is straightforward to include in a rendering system.

Area of Interest

Area of interest (AOI) is a mechanism often used in flight simulators for reducing the rendering load, and involves varying the resolution or quality of the image across the viewing plane **[Cosman90]** **[Mueller95]**. Since the eye has a much higher acuity in the centre of its field of view near the fovea, this is a sensible approach, as long as the centre of detail can be constrained to be along the direction the eye is pointing.

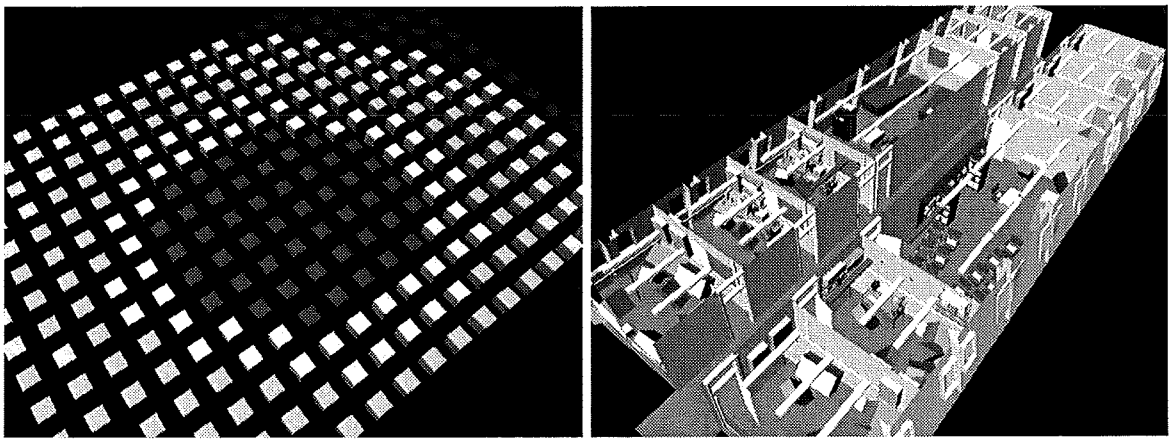


Figure 5.29: Degrading quality in the periphery of the image, for a test scene where the objects are coloured according to the accuracy at which they are used *(left)*, and the same effect for a more realistic model *(right)*.

This can be simulated on a fixed resolution image by smoothly reducing the quality towards the edges of the image. It has the advantage of allowing more detail to be drawn in the centre of the image where it is assumed that the viewer is concentrating. Figure 5.29 shows the results of this on two scenes. The implementation allows any number of **screen lenses** to be placed on the screen, each of which can have a specified radius r_i and drop-off factor p_i. The quality threshold is then scaled by a factor

$$1 + (f_i - 1) \cdot (\cos(d_i/r_i))^{p_i} \tag{5.18}$$

where d is the distance from the pixel to the centre of lens i and f_i is the strength of the lens in the range $(0, \infty)$. Placing lenses at different screen locations can be used to emphasise certain areas of images as in a similar manner to **[Strothotte94]**.

Detail Lenses

This idea can be extended naturally into 3D, where volumetric **detail lenses** can be used to control quality in a manner similar to **[Cignoni94]**. These specify a volume in world-space inside which a quality modification is made and can be optionally smoothed off towards the edges using Equation 5.18. The quality control variables are modified for objects within these regions, either increasing or decreasing quality. Calculating inclusion should be rapid, using primitives such as spheres or axis-aligned boxes as the boundaries of the lenses (Figure 5.30).

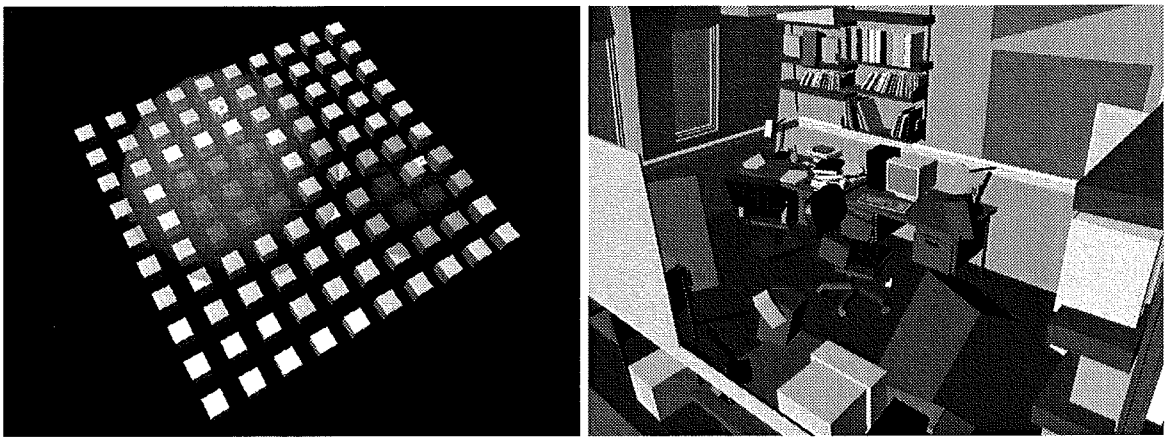


Figure 5.30: Examples of two detail lenses highlighted by transparent spheres, one enhancing detail and the other reducing it in a test scene *(left)*, and a single lens in a more realistic environment, visible by its effect on the objects towards the far corner of the room *(right)*.

One immediate application is in an editor for complex models. By placing detail lenses around objects that are being edited, they can be positioned accurately with respect to the other objects in the model, but maintaining an interactive frame-rate on low-powered machines by keeping the rest of the scene at a low quality level (Figure 5.31).

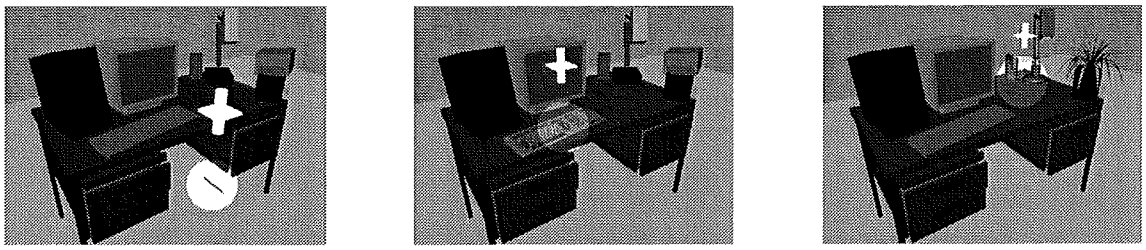


Figure 5.31: A sequence of images from a simulation of an editor. The pencil (highlighted) is moved using the manipulator above it, and the detail lens around it allows accurate placement while maintaining interactive frame-rates.

Depth Exaggeration

An interesting variation on the control of detail is to exaggerate the effects of depth, reducing the detail further than is geometrically required to enhance users' perception of depth. This is easy to achieve for example by modifying Equation 5.10 to

$$F_r(R) = 1/r^4 \tag{5.19}$$

The effect is to no longer present an image of a uniform quality, but one where quality is lower in the distance.

5.6 Summary

This chapter has described how to modify two common types of rendering system, ray-tracing and Z-buffering, to use approximation hierarchies and identify a subset of the model that is needed for the current task. A new framework for quality control criteria has been introduced and two criteria which are easy to use and understand, and also efficient to compute, have been demonstrated. In combination with visibility computations, these provide a means of restricting the model subset required for generating an image to a small localised subgraph of an approximation hierarchy. The next chapter describes how these methods can be placed into a system that can use arbitrarily large and complex models. Other occurrences of approximation within graphics algorithms have been highlighted, and areas indicated in which further research is required to take advantage of approximate processing, including illumination models and fixed-time rendering. The last section has proposed several novel quality control mechanisms for use as an aid to communication — an area in which the benefits are potentially enormous.

Chapter 6

System Design

— In which model managers are described, encapsulating all the elements discussed.

The previous chapters have described approximation and replacement schemes that allow a large and complex database to be handled by accessing only a small portion of the data. This chapter examines the practical consequences of using a model subset, and describes how systems can be constructed around a **model manager** that encapsulates the mechanisms introduced in this dissertation in order to handle large and complex models.

The first section introduces the problems the model manager faces when attempting to maintain the model subset requested by an application with evolving requirements. Sections 6.2 and 6.3 contain a brief overview of how two existing systems approach some of the tasks singled out for the model manager. A detailed framework from which a model manager could be constructed is provided in Section 6.4, and Section 6.5 gives details of an implementation based on these ideas.

6.1 The Role of the Model Manager

As introduced at the end of Chapter 2, the basis of the model manager is the separation of model management from the rest of the graphics application (Figure 6.1). The model manager encapsulates several tasks that are usually left to graphics applications, as well as providing some new features that will enable systems to scale with the increasing complexity of databases and their wide-area distribution.

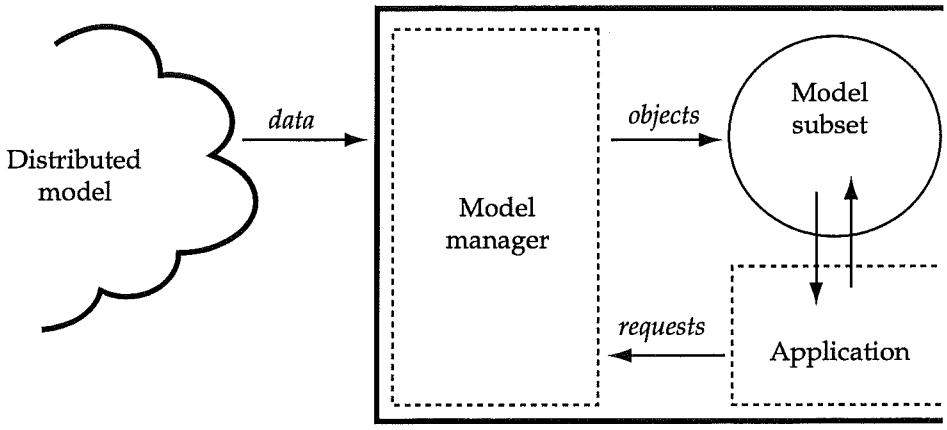


Figure 6.1: The position of the model manager in the system.

The basic task is to manage a **partial model** which in this case will be a representative subgraph of an approximation hierarchy. The problems associated with extracting a partial model from an approximation hierarchy are discussed first, after which the major requirements of the application are identified. These are then examined in the practical situation where limited resources restrict the ability of the model manager to fulfil them. Finally, the cost of introducing a model manager is assessed, and compared to other techniques for achieving similar results.

6.1.1 Partial Models

In Chapter 2, partial models, or model subsets, were introduced as the only way in which arbitrarily large models can be handled. Being able to make use of these subsets relies on the presence of approximations or alternative representations, so that a **valid** subset exists containing *some* drawable representation for *every* object in the scene. The concept of an approximation hierarchy has been identified as a suitable means of holding this information since it has the essential property that it is possible to obtain a valid representation of the model rapidly from this structure.

Physically, the model will only be accessible by retrieving **segments** of the approximation hierarchy at once, each likely to contain many nodes — often more than are immediately wanted (Figure 6.2). As well as reducing overheads, this may introduce performance penalties which will be discussed in Section 6.4.2.

The partial database model is a result of taking a practical approach to large databases. Ideally the application would be unaware that it was dealing with a subset, and would be free to access any part of the model as it wished. The model manager should thus support this level of access as best it can and maintain the illusion of a **virtual model**, containing data only in the form the application requires.

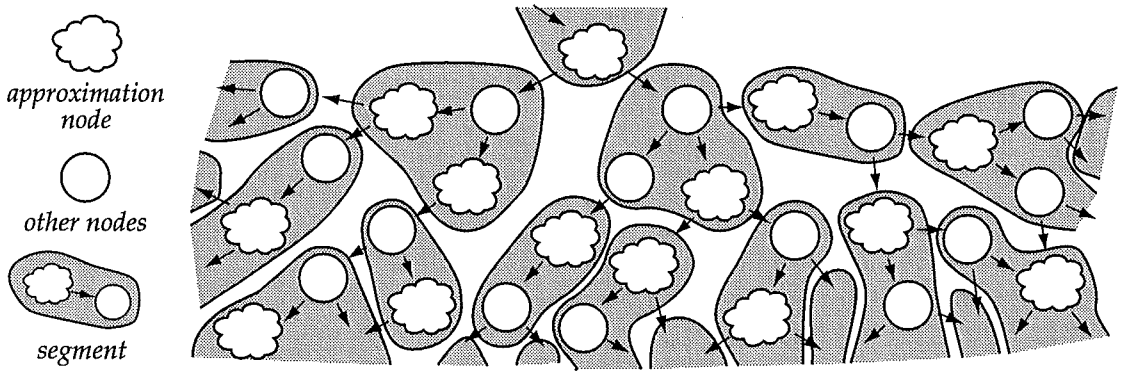


Figure 6.2: Part of an approximation hierarchy split up into a number of segments, the unit of data retrieval.

The scheme described here is independent of the database storage system used to obtain data, although implementations should make use of all benefits that are available to them. The requirements placed on the model manager are very restricted and different from the general database context, but could take advantage of support for data retrieval — Section 6.1.4 compares the tasks of the model manager with ideas from persistent databases.

6.1.2 Application Requirements

The model manager should aim to fulfil the following application requirements:

- The application should always have access to a *valid* model.
- The application should be free to request *any* data at *any* point in time.
- The model manager should return objects as rapidly as possible after a request.
- The application should receive data in the form in which it is required.
- The data should remain freely accessible to the application once it has received it.

With real-time responses required and finite memory and communication bandwidth, it will not be possible to totally maintain the virtual model illusion. Nevertheless, these requirements provide a useful starting point when designing a model manager.

The performance of the manager when responding to requests is primarily in the hands of the database or filing system from which it retrieves its data — subjects which are not covered by this dissertation — although in the case where data is already in memory, a fast response is essential. Format and type conversion, and validity, are both incorporated into the approximation hierarchy using the mechanisms described in earlier chapters.

Once data has been requested and then loaded into memory, the final responsibility of the model manager is removing data when it is no longer required. When browsing through a large model, the number of data items requested will steadily increase, but many of the

objects loaded earlier will not be used again and should be removed to free space for new objects. This is the main obstacle in giving the application *complete* freedom to use the data once it is loaded.

In order to be able to solve the problems of a particular application efficiently, it will be necessary to tailor the model manager to each application. All model managers will perform essentially the same task, but the details of how these are carried out and how conflicting requirements are resolved might change from application to application. The interface and control mechanisms between the application and the manager provide the greatest scope for optimisation. For instance, the model manager could place data directly into the memory structures used by the application. Similarly, the conversion routines required to convert data, and the types of data that can be handled, are application specific. It should be possible, however, to construct most of a model manager from a standard library of components.

6.1.3 Time-Varying Requirements

Compromises have to be made when there are not enough resources to service all the requests made for data. For a static subset, given enough time it should be possible to retrieve all the data, but in a dynamic application, the partial subset required changes with time, so that more data will continually be requested, while older data is no longer needed, and the model manager must adaptively refine the set of objects it manages.

The model manager acts like a **cache** system for objects, and some concepts from memory management can be applied. If the manager **pre-fetches** objects it believes are about to be needed by the application, then when the actual demands are made the data can be delivered immediately. When the manager is not running under maximum load, the cost of retrieving the data is absorbed naturally. The penalty is that time may be wasted on fetching data that is never used.

For general data management, deciding which objects to pre-fetch is an impossible problem to solve, but in the structured data environment described, the manager is greatly helped by many forms of coherence. The key observation about segment retrieval is that the application only knows about a restricted set of objects — those that are already in memory-resident and those to which they are immediately connected (Figure 6.3). Change can only occur by loading objects from the **fringe** set containing the objects that have been *referenced* but not loaded.

While substantially restricting the search, this is not all that can be achieved. Since the data represents 3D spatial information, there is a large amount of coherence that can be exploited. Used in conjunction with the temporal coherence of consecutive viewpoints, this enables potential changes to the set of required objects to be predicted. Much of this is guesswork based around visibility coherence (Section 2.1), however advantage can also be taken of a significant amount of **complexity coherence**, since the quality at which objects are needed is also likely to vary smoothly.

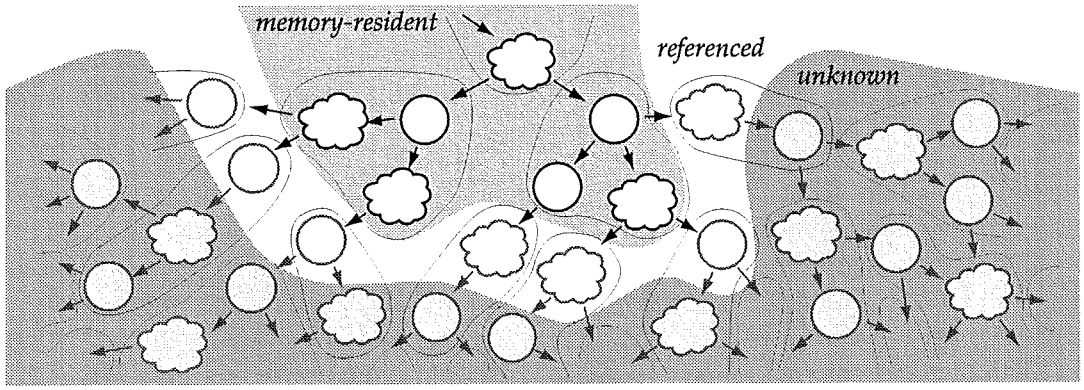


Figure 6.3: The application only knows about a restricted set of segments on the fringe of the currently loaded model.

It may be better to leave the calculation of coherence information to the application which can use higher-level knowledge of the data that is being manipulated. This can then be passed on to the model manager through explicit requests for data, or more usefully, by setting priorities for un-fetched data to be used as a basis for pre-fetching by the manager. It may be the case, however, that a particular model manager has access to the information required to do such work on its own, just requiring feedback from the application about the viewer's position and current motion.

6.1.4 Costs and Alternatives

The majority of graphics applications are based around the "whole model" approach of reading all the data into memory when the application is started. If there is enough time and memory to do this, then the application has guaranteed access to the whole model, but if only a small proportion of the model is actually viewed then time and memory are wasted. For complex models, this is increasingly likely, and costs soon become prohibitive.

Using a model manager, loading is done **lazily**, *i.e.* when (or ideally just before) the data is actually needed. This exchanges a lengthy delay when the application starts for a smaller delay during execution. The memory requirements can be significantly smaller if not all of the model is accessed, and the total time spent loading data will be less. While delays during use can be a nuisance, they can be minimised through a good pre-fetching scheme, and because of the use of hierarchical approximations, never prevent a frame being drawn, since another representation is always available for all objects.

The main penalty with such a system is the additional information that is required in order to perform the management tasks efficiently, and the time overheads of maintaining these structures. A good pre-fetch scheme might require a large structure maintaining visibility information in order to make its decisions. The cost of this has to be balanced against

the waste that might occur due to bad pre-fetching. As a general rule, the form of an approximation hierarchy should ensure that the size of data structures needed in memory is proportional to the size of the partial model loaded (together with the fringe set) but independent of the size of the *whole* model.

There is plenty of opportunity for multi-tasking or multi-processing in such a system. The obvious break is between the application and the model manager, although the tasks within the model manager could also be separated. An example configuration is a two-processor system where one processor is dedicated to the application while the other deals with model manager tasks (Figure 6.4). In order to avoid wasted data transfer, shared memory would enable the model manager to place fetched data directly where required by the application.

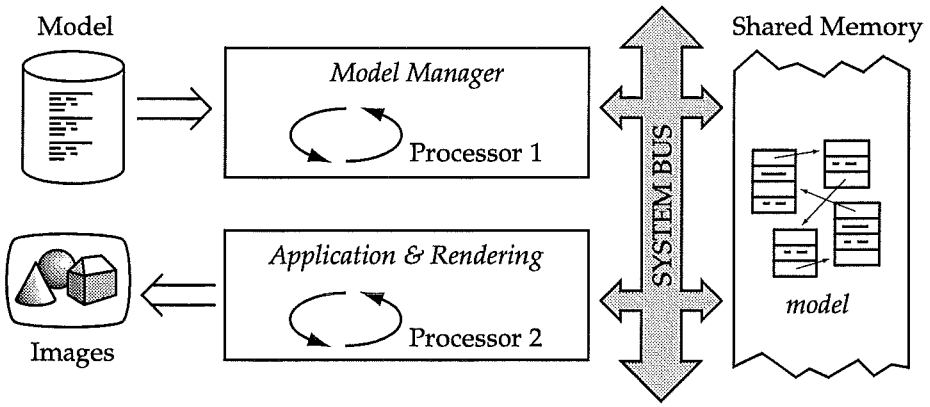


Figure 6.4: A multi-processor configuration with the application and model manager running in parallel using shared memory.

Virtual memory systems swap data to and from disc to simulate a large address space with finite memory and are similar to model managers in some respects. Using virtual memory management to control access to model data has a number of disadvantages. Since paging is performed in fixed size blocks without any explicit knowledge of the contents of the block, the application may find that the data it requires is not immediately available, and has no guarantees about validity. In addition, a least recently used (LRU) page scheme may not be the best that can be managed for determining which objects are not required in the near future.

The system described here shares several aims with **persistent programming** [**Atkinson83**] (*e.g.* PC++ [**Wu93**] or Arjuna [**Dixon89**]). The separate model is formed from a collection of persistent objects, and the model manager encapsulates data migrating mechanisms, relieving the application of having to read and write data explicitly. The approach in this dissertation differs in two respects. Firstly, persistence only applies to a very restricted class of data items rather than arbitrary data-structures, and the access patterns to this data are much more predictable. Secondly, real-time response is achieved through the use of ap-

proximation hierarchies, multiple representations of data and pre-fetching — the application no longer depends on having access to exactly the data required, or utilising methods such as remote object invocation. An interesting area of research is the extension of database systems to incorporate multi-resolution data retrieval [**Read92**] [**Stonebraker93**] and should incorporate the notions of type conversion as well as standard type checking. As well as these issues, it is important that data does not have to exist on a dedicated storage system or in a restricted form. The implementation of locking, recovery and consistency checking is not necessarily required in the existing system, although there are applications where such controls might be important. Finally, efficiency is an important consideration, and the access methods described later in this chapter are based around the goal of removing all but the most essential overheads.

Several general graphics systems attempt to provide "model management" facilities. These are usually aimed at either loading data, or storing it for use (*e.g.* PHIGS [**Foley90**] or IRIS Performer [**Rohlf94**]) or are special case solutions such as the architectural system described in the following section or terrain visualisation and flight simulators (*e.g.* [**Falby93**] [**Mueller95**]). Problems that often arise are due either to inflexibility, or the isolation of data from the application. In particular, it is common that such systems attempt to encapsulate the model *and* the rendering, providing insufficient control for applications which then keep second versions of the data for their manipulation and have to keep updating the system copies. The following two sections describe how two existing systems implement some of the features introduced here.

6.2 Case Study — A Building Walk-through System

The architectural walk-through system described in [**Funkhouser92**] uses an object-caching mechanism in order to operate within a fixed memory budget. Architectural environments are densely occluded so that only a small subset of the model is usually visible at any one time, and the methods employed are mainly based on visibility prediction. The accuracy of the visibility prediction algorithm is increased due to the limited set of objects that can be seen from a viewpoint and also because the motion of the viewer is restricted.

6.2.1 Object Management

The algorithm is based on exploiting visibility coherence, and an expensive pre-processing stage creates several data structures from the original model that will allow efficient indexing into the model during the walk-through phase.

- The architectural model is split into **cells**, which are selected so that their boundaries are largely opaque [**Airey90**]. Objects are stored in the cells in which they are positioned, producing a two-tier hierarchy of cells containing lists of objects (Figure 6.5).

- The non-opaque regions on the boundaries are termed **portals**. These are processed in the portal enumeration step to form an **adjacency graph** connecting the individual cells by shared portals.
- A potential visible set calculation [**Teller93**] is performed to construct a **cell-to-cell** visibility graph, indicating all the cells that can be seen from some position within each cell, and a **cell-to-object** graph listing objects that are visible (Figure 6.6).

Currently the visibility calculations are restricted to axis-aligned splitting planes, together with axis-aligned rectangular portals. Visibility between cells is computed using a depth first search of the adjacency graph, limited by considering sight-lines through portal sequences. The distinction between cells and objects is a product of the densely occluded environment — the cells encapsulate the major visibility blocking elements, and objects and other cells have their visibility stored relative to each cell.

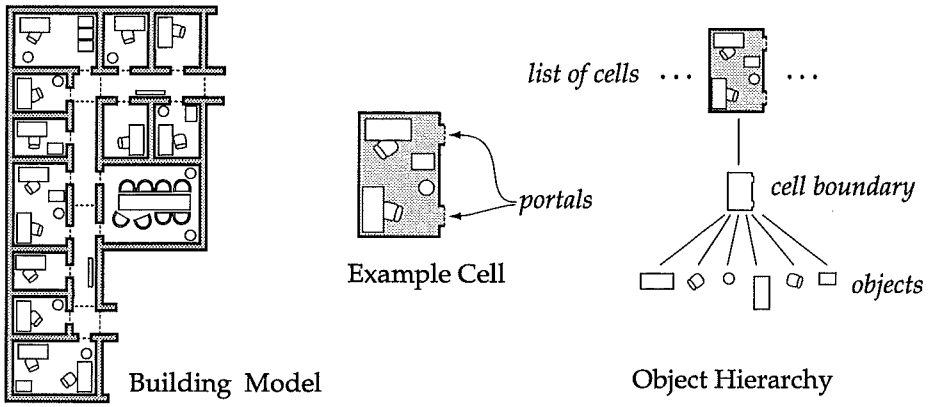


Figure 6.5: Illustration of the cell structure and model hierarchy.

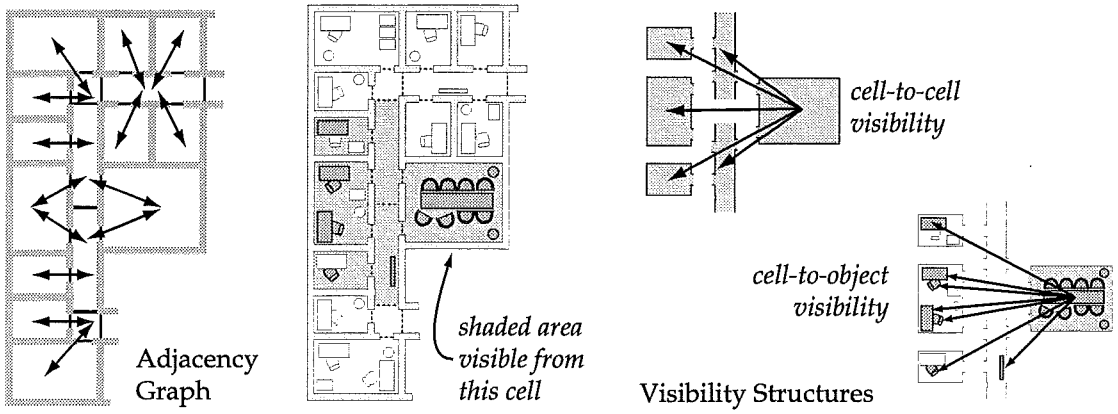


Figure 6.6: Illustration of the visibility structures constructed to provide efficient indexing into the model.

During interactive walk-through, a cell and object cache is managed in a separate process. The pre-fetching strategy is based on viewpoint prediction for the next N frames, proceeding as follows:

- Cell-to-cell and cell-to-object visibilities are further restricted to **eye-to-cell** and **eye-to-object** visibilities based on the current eye position within the cell and the viewing frustum.
- Cells are assigned weights based on the minimum number of frames it would take the user to enter that cell given the constraints on movement rates.
- The cell-to-cell visibility information is then used to assign priorities to all cells that might become visible in N frames time. Objects are processed as well, and LODs are picked based on the distance from the viewer at which they would appear.
- This prioritised look-ahead set is then given to the cache management process. A **read set** and a **release set** are constructed from the requests based on priority and the cells currently loaded.
- The read set is processed in priority order until all loading that can be achieved in the current frame-time is performed, at which point no more data is transferred until the requests for the next frame are received.

The read set is formed by adding objects in order of priority from the look-ahead list. Objects which are already resident in memory are not added to the read set. LODs are always loaded from lowest to highest detail for each object.

Objects are released on a LRU basis. Objects are kept ordered by the time when they might potentially become visible. When an object already in memory is noticed in the look-ahead set, it is moved up to the front of the queue.

6.2.2 Performance

A sophisticated LOD management technique, described in Section 5.5.4, is employed within the rendering subsystem in order to maintain a roughly constant frame rate. The use of LODs is limited, with just two or three representations for each individual object, created either by hand, or by manually adjusting values in a parametric design package. Since there is no hierarchy of objects, visibility information is computed and stored on a per-object basis, and the calculation is therefore linear in the number of objects in a given cell. Although efficient for the scale of detail present in the example model, as the detail increases, the lack of hierarchy would become prohibitive.

When all the objects that are required for a frame are not loaded in time, either due to slow disc accesses or unusual viewer movement, users find that it is more acceptable to simply render the scene with the objects missing, rather than stall the graphics application in order to wait for the objects to be loaded. While preferable to loss of control for small objects, when an object such as a cell boundary (*i.e.* a wall) is absent, the effect is disorienting. When a different LOD is in memory for an object that is required, then that can be used as a re-

placement. If a deeper approximation hierarchy exists, then this situation is less likely to arise since it should always be the case that some representation of the object is available.

The success of this system is largely dependent on the type of model in use and the large amount of pre-computation performed. This results in some large data structures that are the basis of the real-time management algorithms. In addition, the database is processed and stored in a manner that facilitates rapid loading of groups of related data. In the example described in [**Funkhouser95**], the total model size is roughly 350Mbytes, from which a 16Mbyte object cache is maintained. The total precomputation information is also about 16Mbytes. With larger or more detailed models, or environments without such restricted visibility, the size of this data will increase. In order to cope with this situation, the information must be stored locally rather than globally so that only the portion that is needed immediately is memory-resident. It may also be the case that a different balance might be struck between rapid visibility results and amount of precomputation.

6.3 Case Study — VRML 1.0 Browsers

The **virtual reality modelling language** (VRML) [**Bell95**] is a new standard for describing models over the Internet, and its authors hope that it will do for 3D modelling what HTML [**HTML**] has done for textual and graphical resources. The language itself is based on a subset of the Open Inventor file format [**Wernecke94**], together with two extensions for fetching data across the World Wide Web (WWW). VRML inherits a fairly flexible notion of a hierarchical scene graph composed of nodes, which include level of detail nodes based on the distance of the object from the observer.

The two network nodes are:

- **WWWAnchor:** This node is the equivalent of the textual hyper-link in HTML, and when the object it contains is activated, for example by a mouse click, it instructs the browser to jump to the document that is referenced in this node.
- **WWWInline:** The inline node is equivalent to including the referenced VRML file in the original scene, using a **universal resource locater** (URL) to reference a file from anywhere on the WWW. Its purpose is to allow files to be included compactly so that scenes can be assembled from predefined component parts.

Of these two nodes, the latter, in combination with the level of detail node, is of the most interest.

6.3.1 Distributed Models

The purpose of the inline node is to make VRML scalable by enabling the worlds that are created to be distributed across many different machines. This not only distributes the disc space required to store the worlds, but also the server time required to transfer them. Efficiency can be improved if browsers support the pre-fetching of inlined data and also the caching of inlines that might be used multiple times.

Additional fields are available in the WWWinline nodes, as well as the location of the data to be fetched, describing the size and centre of the object's bounding box. These are typically used by a browser to enable incomplete scenes to be processed before all the objects have been fetched across slow network links, and can also be used as simple approximations. Users can interact with a scene in the meantime, and move around inside it. Loading can be carried out in a multi-threaded manner largely independently from viewing. Many HTML browsers use similar information to layout and display incomplete pages while they are still being downloaded, greatly increasing the productivity of the user.

6.3.2 Use of Levels of Detail

Currently, many VRML files are fairly large and take a while to download (*e.g.* 600 Kbytes might take 2 minutes to transfer). While the WWWinline node is intended to assist in decomposing files into manageable portions, it is not often used in this way. Similarly, the LOD node is intended to save not only rendering speed but, in conjunction with the inline nodes, to save time by reducing the latency between the request for a scene and the time of display.

The LOD node would typically contain two or three children. The first would have some explicit simple geometry, the second would be an inline node linked to a moderately detailed representation of the object, and the final one would be an inline node to the most complex version of the object (Figure 6.7). The two inline nodes could furthermore link to different sites with different expected access times. Using different LODs also enables faster movement on less powerful hardware.

Issues that are still being addressed **[Bell95]** include

- compression techniques for increasing transfer speeds
- the order in which nodes are fetched over slow links
- the use of object caches either on the network or as a CD-ROM object archive distributed to users
- the use of universal resource names (URNs) rather than URLs to facilitate object caches by separating name information from physical location
- coping with remote data that changes, in particular for dynamic shared environments (*e.g.* **[Broll95]**).

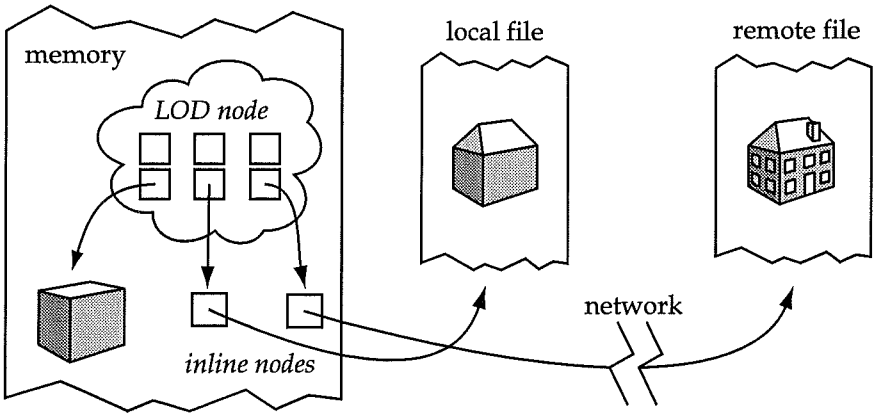


Figure 6.7: A typical use of the WWWInline node in conjunction with a LOD node.

Apart from some of these issues, and problems with many of the current scenes available, VRML looks like a reasonable candidate for constructing complex models. The one drawback is that it is a scene language specification, and as such will place limits on what individuals wish to describe. It is fairly flexible and extensible, nevertheless other formats will continue to coexist. One solution is to expand the range of converters available and automate their use from within VRML.

6.4 Model Manager Design

This section examines the internal components from which model managers can be constructed, and the implications for model construction, particularly segmentation. The components actually chosen will depend on the application. For example, a model manager which is associated with a ray-tracer will be much more flexible in waiting times, and more determined to retrieve the data requested than one associated with a real-time system that must produce a frame every 1/25th of a second.

6.4.1 Components

The model manager has three main functional tasks (Figure 6.8):

Request Processing: It will not always be possible to respond to all requests as quickly as they can be issued, therefore it is necessary to maintain a **request queue**. If priorities are associated with requests, they can be ordered. Enough information needs to be stored to be able to identify and service the request.

Release Processing: If objects are to be removed from memory when no longer needed then these need to be identified for removal by placing them on a **release queue**. Again removal might not be immediate, and if the system has only a limited amount of memory available it may be necessary to remove objects that are still required, in which case segments can be given a removal priority.

Type and Format Conversion: When data is obtained from the database it may need to processed before passing it to the application. This might involve format and type conversions or approximations, as well as the processing of the data to obtain any information required by the model manager.

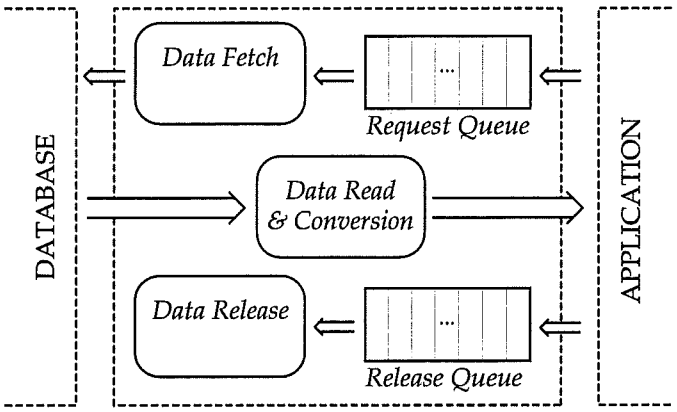


Figure 6.8: The main components in a model manager and relationships between them.

The approximation hierarchy is processed in segments, and the model manager needs to keep a **segment record** containing enough information to retrieve a segment from the database, isolating the location and fetching information from the application. It is useful to maintain these records not just for data that has been requested, but also for all segments resident in memory since it provides a convenient handle for the data and facilitates segment removal. The record also needs to contain an indication of the state of the segment — whether it is memory-resident, has been requested, or is currently being fetched or processed by the manager.

For **referenced** objects which have not yet been loaded, the segment record acts as a placeholder, or proxy, for the real data enabling multiple requests for the same segment to be identified rapidly and hence avoided. If an index is kept for these records, then it is possible to use them for checking that the data requested has not already been loaded from another part of the model.

Segment records form the major memory overhead for the manager and the number and size of these should be minimised. The number is determined by the average segment size, or model **granularity**, and the number of segments resident or referenced. Much of the size

of each record is devoted to the location information used in its retrieval. Once the data has been retrieved, it might be possible to discard this information, but should it be removed from memory and requested again, the location information would be required.

6.4.2 Model Segmentation

While the model can be split into segments in any manner, careful choice of segment boundaries helps the model manager to function efficiently. The model manager only passes data to the application if it is valid, and a whole segment is valid only if all the approximation nodes it contains have at least one representation loaded. If, when a segment is loaded, it is valid, then it can be passed immediately to the application; otherwise, further processing is necessary, either to load other segments to create a valid subgraph through a closure operation (Figure 6.9 *(left)*), or to attempt to truncate the segment to a valid subgraph itself (Figure 6.9 *(right)*).

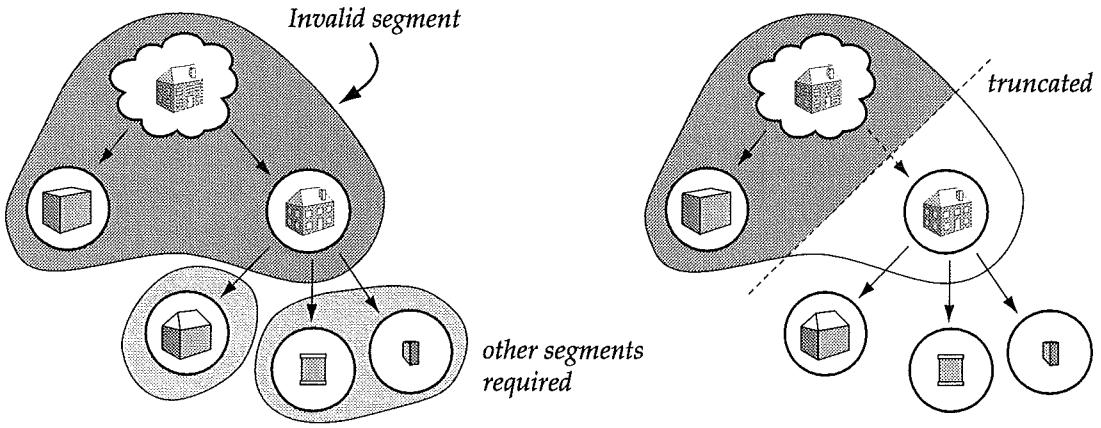


Figure 6.9: An invalid segment either requires further segments to be loaded before it becomes usable *(left)*, or for the segment to be truncated to produce a valid subgraph *(right)*.

Since low-quality representations are invariably required before high-quality ones, segments typically contain approximation nodes with just the lowest quality representation of an object. The higher quality approximations, if of appreciable size, are stored in separate segments (Figure 6.10) which may never need to be loaded. This enables the low-quality object to be displayed rapidly, without having to wait for a higher-quality one to be transferred. Segment sizes should not be so small, however, that the overheads of processing them, such as storage for the segment record, grow excessively. The model manager can, of course, transparently package several small segments into a larger one when they are loaded.

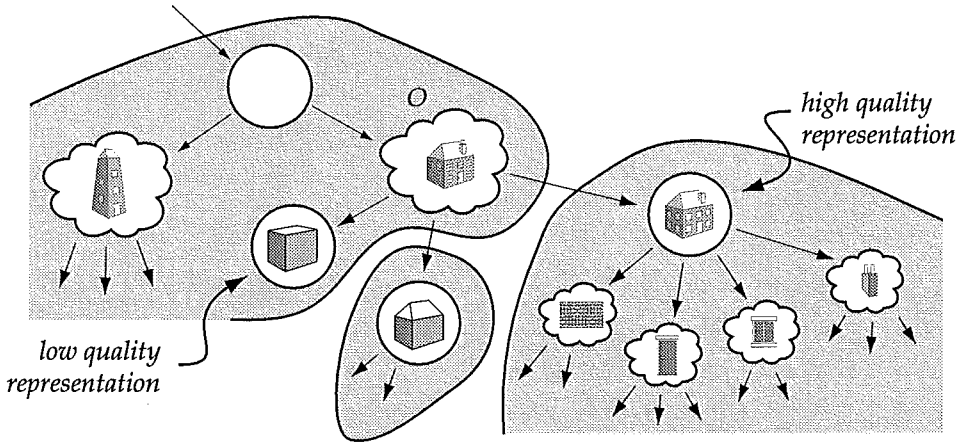


Figure 6.10: A typical segmentation of a part of an approximation hierarchy. The lowest quality representation of object O is in the same segment but higher-quality representations are stored separately.

6.4.3 Problems with Distributed Models

The segmentation strategy advocated in the previous section leads to objects with representations that are physically separated from each other, resulting in the benefits outlined above. This separation can also lead to problems, particularly when one representation of the object is edited. While all representations remain in a local domain, each can be updated at the same time, but if copies of the low quality version have been distributed to the many different segments where the object is used, locating and changing all of these is tedious.

When editing an object, an approximation at a higher level in the hierarchy which is based on this, and other objects, may also need to be recomputed to avoid **representation inconsistency** (Figure 6.11) — a common problem with distributed storage systems. If one representation of an object is modified even in a small way, for example by changing its colour, other representations of the object that are based upon it then become incorrect.

Tackling such issues is an interesting problem in its own right, but a model manager should be designed with such situations in mind. If a version number or time-stamp is stored as part of the model, then it should be simple to *detect* that a representation has changed when it is accessed. Since the manager will have reached this representation through a known path, it can indicate to representations along this path that they may need updating, propagating the information up the hierarchy. The opposite approach is to store information with each segment about other places from which it is referenced so these can be informed of changes in order to maintain consistency in what is in effect a large distributed database.

A further problem that may occur with distributed models is that a segment becomes inaccessible, perhaps because of a network fault, or because the data has been deleted. Approximation hierarchies ensure that there is always a higher-level approximation for the

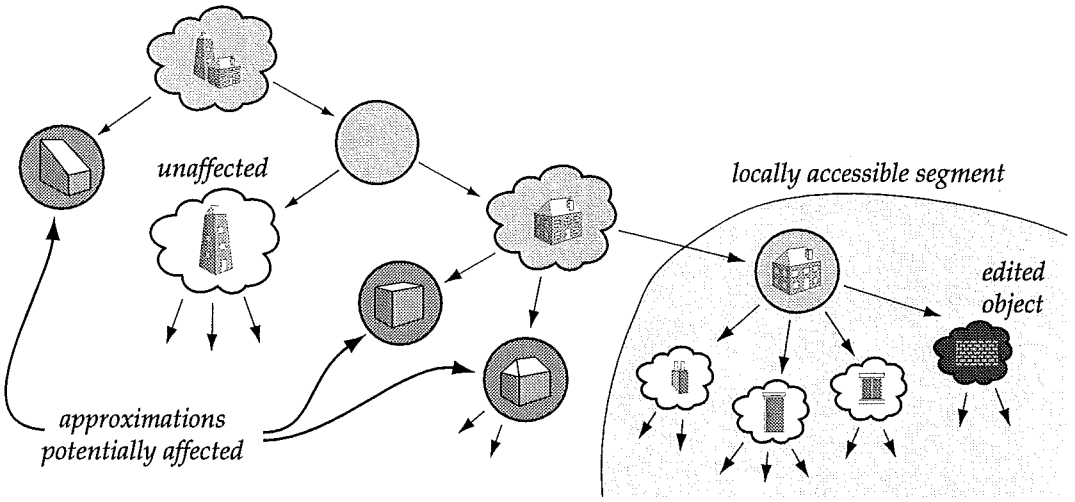


Figure 6.11: Changes to object representations can affect many other nodes in a model to which access may not be directly available.

object. Careful segmentation avoids other objects becoming unusable even though they *are* accessible (Figure 6.12).

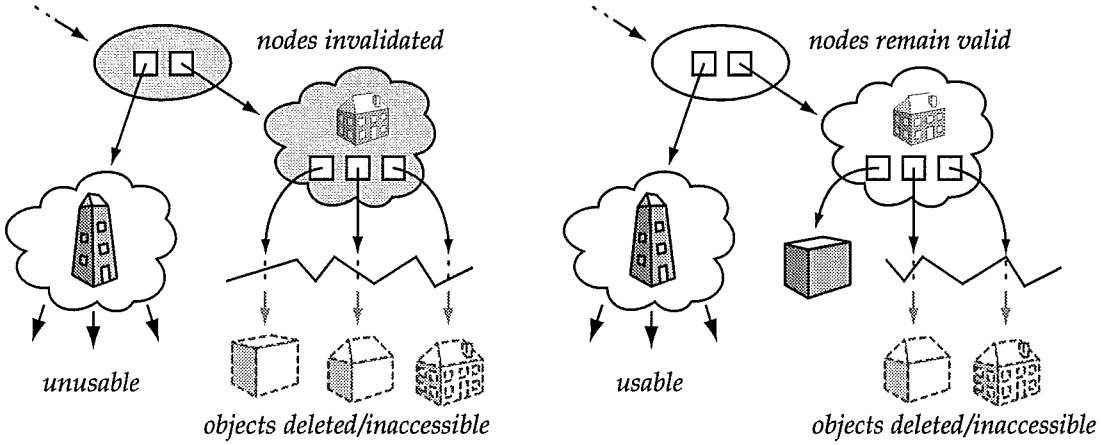


Figure 6.12: Part of a scene hierarchy with broken links where objects are inaccessible, either propagating beyond the affected object *(left)*, or leaving all other branches still usable *(right)*.

6.4.4 Type and Format Decision Making

One important task of the model manager is to isolate format and type differences from the application by automatically converting from different representations. The model man-

ager has a choice of either invoking **external** converters, or calling **internal** routines (Figure 6.13). Conversion routines may also be called, unknown to the manager, by a remote server which is providing the data — in this case the model manager is only aware of an object with several different representations.

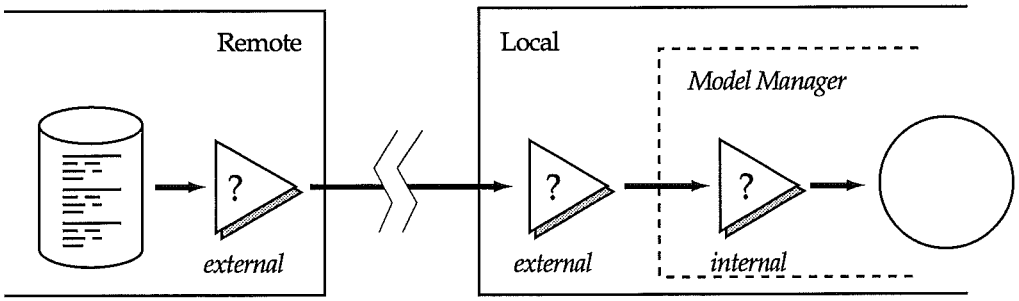


Figure 6.13: The possible stages at which conversion can occur in the path of data from its source to the application.

A route between the source and destination type needs to be chosen from the network of translation paths defined by the conversion routines available (Figure 6.14). The destination type or types are known since they are defined by the application and the manager is constructed with this knowledge mind. The last routine in any conversion path is responsible for leaving the data in the internal format required by the application. Since these types are application specific they are likely to be few in number.

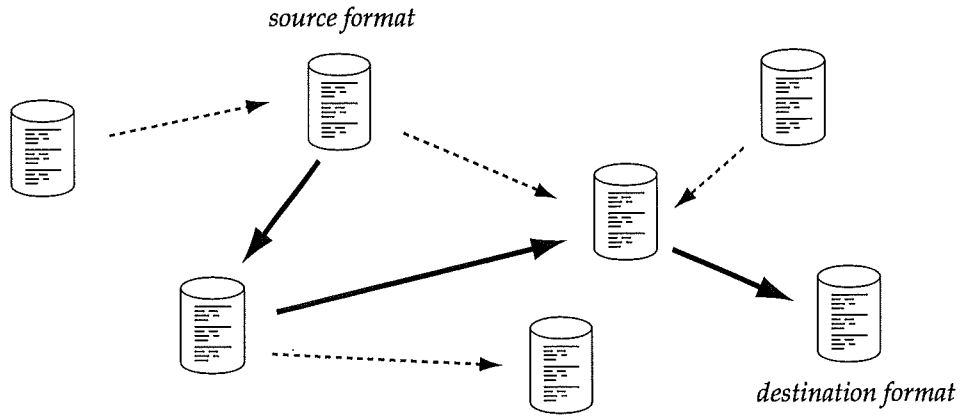


Figure 6.14: The network of translation paths available between types of data, with a possible path between the shown source and destination types highlighted.

There may also be a choice involved when asking for data from the storage system, since different representations in different formats may be directly available, adding more possible paths to the translation network. The cost of possible paths depends on several factors

including the estimated time cost to retrieve the data in the chosen format and the times to run conversion routines on it, the quality of the input and output of these routines, and the desirability of the destination types reached. The relative importance and measurability of these factors depends on the context within which the application is executing and is one point where model managers should be optimised for the application they service.

The case where there is *no* path between source and destination type needs to be accounted for. This appears to the model manager as if the object is inaccessible, and it must fall back on a different representation or a higher-level approximation in the same way as illustrated in Figure 6.12.

6.5 An Example Implementation

To illustrate how a working system incorporating the ideas presented in this chapter can be developed, this section describes a simple implementation of a model manager. The application is written using the Open Inventor [**Wernecke94**] toolkit on a Silicon Graphics Indy XZ uni-processor workstation [**SGI94**]. Data is retrieved from a networked UNIX file system rather than any dedicated database and any modifications to the model are not written back to disc. Objects may also be fetched from remote locations using the WWW retrieval mechanism. The model structure, both external and internal, is described first, followed by the supporting data structures within the manager. Details of the **type manager** used to make decisions about conversion routes are given in Section 6.5.4, and finally the different configuration options are compared and the operation of the system analysed.

6.5.1 The Model and Model Subset

Models can be composed from a number of different sources. These include existing objects modelled by a number of different techniques and stored in several different formats, together with objects that are generated on-the-fly both internally and externally to the application. All segments are stored in an external format to maintain generality, in contrast to the internal format used by [**Funkhouser92**] which enables rapid swapping of data segments.

Various conversion utilities and parsers are available to the model manager for dealing with objects specified using different languages, or for objects of types that cannot be handled directly by Open Inventor. File types are currently identified through context and filename extensions.

In the application, the model subset is managed using the Open Inventor scene graph structures, together with a few additional nodes that provide lazy object-loading features and interface to the model manager. The unit of data loading is generally the file, and lazy nodes usually contain a file reference (or a URL) to specify the location of data.

In some circumstances the model manager may control the amount of data that is loaded in response to a request. Since segment records are created *only* when loading a portion of the model, the model manager can decide whether to load data or just create a reference. Splitting an incoming segment relies on being able to continue the retrieval from that point later, and is therefore restricted to breaks between files. Segments can also be combined into a larger segment under a single corresponding segment record, typically in order to make a segment valid.

6.5.2 Internal Structures

The model manager's internal structures are kept as small as possible in order to minimise memory overheads arising from the management process. A segment record is stored for each portion of the model that is loaded through the model manager. The record contains location information identifying from where the segment can be fetched, a pointer to the segment if it has been loaded, and a status descriptor.

As well as the request queue and release queue described in Section 6.4 two other lists of segment records are kept so that all records are always in one of four places (Figure 6.15). The **named list** contains all new records for segments that are not loaded and have not yet been requested and the **fetch list** contains all segments that are in the process of being fetched.

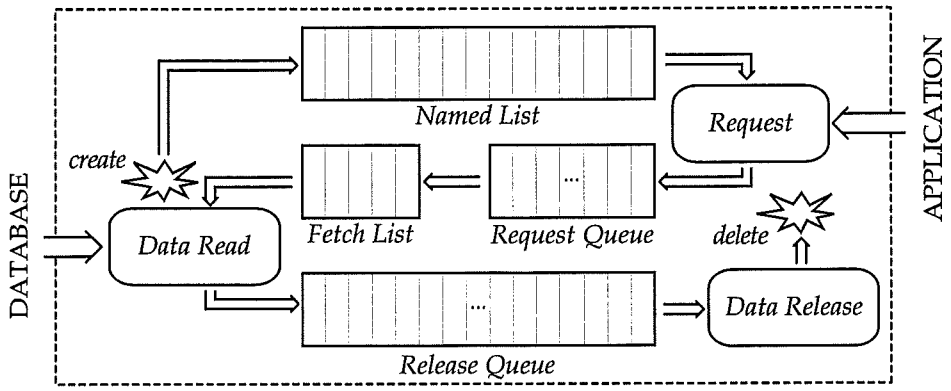


Figure 6.15: The four lists of object records and the paths of segment records within the model manager.

The best way to describe the processes involved is to follow the lifetime of one segment S. When its parent P is loaded into memory, the reference to S is parsed and a new segment record R_S is created for S and placed in the named list. The application is given a pointer to R_S to store in P. At some point, the application may need a higher quality representation of P and will make a request for segment S by calling the model manager request routine, passing it R_S. The model manager checks the status of the segment record. If the object has been loaded in the meantime, the address of the data is passed back immediately to

the application, otherwise the application proceeds knowing that it will have to wait for the data to arrive. If the object is already in the fetch list then nothing further needs to be done. If it is in the request queue, the priority can be modified to reflect the new needs of the application. Otherwise, R_S can be removed and placed in the appropriate position in the request queue.

When processor time is allocated (and all higher priority requests have been processed) the model manager moves R_S from the request queue on to the fetch list and initiates the fetching routines. Loading and parsing the data involves creating a scene graph for the application; when complete, R_S is placed on the release queue containing all the segments that are available to the application. Records are removed from the tail of the release queue by the model manager when it is about to remove the segment from memory.

Each of these structures is currently stored as a doubly-linked list, with pointers and priority value stored as part of the segment record, enabling an object record to be manipulated immediately without requiring any further information. The request queue is maintained sorted in order of priority to enable the removal of the highest priority request, as is the release queue so that the least important segments can be identified.

6.5.3 Model Maintenance

Validity is ensured when segments are loaded by checking the incoming data and only giving the application access to valid branches. An invalid approximation node has its children fetched in order, starting at the lowest quality representation, until it becomes valid. When the model manager is certain that the data is valid, it updates the pointer in the segment record and the data becomes instantly available without the need for any locking in the application. The application uses its handle on the segment record to retrieve the loaded data.

The pre-fetch operation is a convenient means of encapsulating the application's interaction with the model manager. It traverses the scene graph, making requests for segments it anticipates will be needed soon and changing the priorities of objects that are currently loaded. A simple pre-fetching strategy uses the current quality setting as a basis for predicting that objects with similar qualities are likely to be required next. Advantage could be taken of any visibility structures available, but visibility prediction is not currently used except to limit the traversal to non-culled branches.

Pre-fetching can either be applied between frames before any rendering is performed; or interleaved with rendering and applied locally to sections of the graph as they are processed. If pre-fetching is limited to only restricted branches of the hierarchy, then an initial pass by the model manager through the segment records can be used to reduce the priority of the segments *not* traversed during pre-fetching. Segment requests are assigned a priority value based on the quality of the objects contained projected into screen-space. Segments are thus fetched in decreasing order of the quality that the objects that they contain will add to the

image. Rapid fluctuations in priority value can be smoothed using a running average.

Segment removal is deemed necessary when the size of the model subset in memory grows too large. The priority values associated with segment records in the release queue is used as a basis for choosing which segments to remove. The model manager removes low priority segments in order to service higher priority requests, but then has to make sure that the application no longer attempts to access the resulting invalid branches. This can be enforced by constraining the application always to check the segment record before using data. The application can also indicate to the manager that certain representations of some objects are no longer needed, after which they can be removed by the manager. This can be incorporated into the pre-fetching operation by removing branches from consideration which contain data earmarked by the manager for removal. Only when this has been done does the application give the model manager permission to actually remove the data itself. Data which has taken a significant time to transfer can be stored on a local file system (acting like a second-level cache) at removal time in case it is required again in the near future.

6.5.4 Object Approximation and Type Conversion

Incoming data is handled by a **type manager** incorporating a number of modules capable of processing input from different formats through external converters. It also encapsulates routines capable of approximating an object of one type by objects of other types. As well as reading data in a number of scene description formats, there are modules which can invoke decompression programs to handle compressed data. Where possible, external format converters are chained together into a UNIX pipe to avoid reading, processing and saving temporary files.

Information is stored for each format conversion routine, describing the source and destination types, and cost and quality factors. Decision making is based on a simple static cost for each conversion method, and is encoded in a **conversion table** indexed by input format. This describes for each input type the first conversion method that should be applied. Subsequent stages in the conversion are determined by looking up the new format in the conversion table (Figure 6.16).

The conversion table is computed incrementally so new types and conversion routines can be introduced dynamically. If a routine is considered for a conversion that is already in the table, a choice needs to be made. For real-time needs, the lowest-cost path would probably be the most appropriate, but in this implementation a highest-benefit path — *i.e.* quality divided by cost — is picked. If an unknown type is encountered and no information can be found regarding it, then the model manager is aware of this and can indicate the unknown object's presence.

Objects described by a higher-level **object meta-file** allow a simple implementation of more abstract object types. The meta-file contains location, type and quality information about a number of representations of an object. This information is stored in the model manager

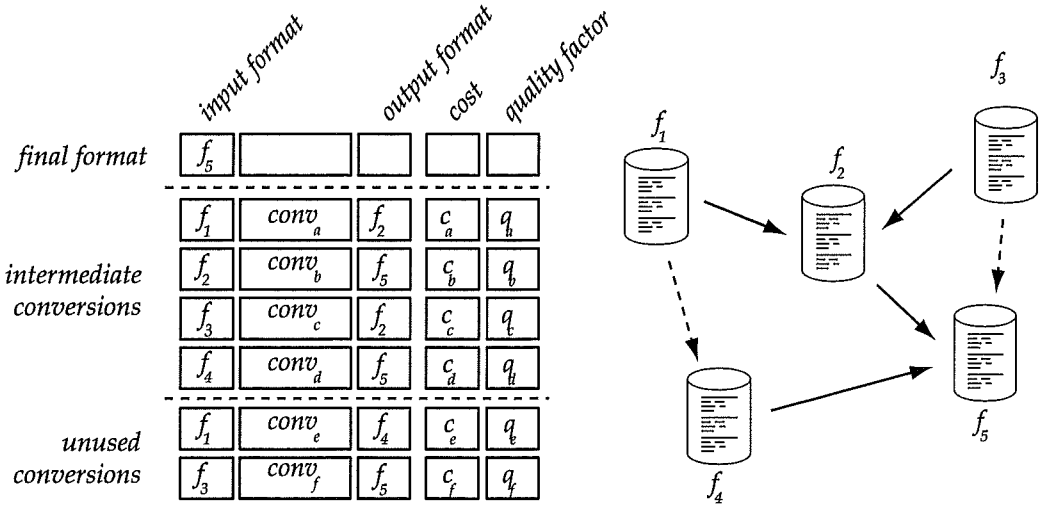


Figure 6.16: Representation of a type conversion graph in an approximation path table.

and is used by the type manager in combination with the information about format conversion routines to choose which representation of an object to fetch, keeping the process separate from the application which just sees an approximation node.

The location of the data as well as its size contribute to the retrieval cost, and this is then combined with the cost of any conversion routines that must be applied. Typically, requests are made for representations of a given quality, and different strategies can be adopted to choose which actual representation to obtain. For a quality-oriented application, a higher quality object should always be fetched if possible, whereas for a real-time system, a least-cost or maximum-benefit approach is preferable.

On the whole, the current system assumes that approximation has been carried out in advance, and that the data is arriving as part of an approximation hierarchy. No approximation routines are incorporated directly in the model manager, although type modules can use external applications to generate approximations for data if required. The only reason for this setup is simplicity. These applications could be linked into the model manager and create data structures straight into memory.

6.5.5 Configuration and Results

The model manager can be configured to run as one or more processes separate from the application using shared memory, or in the same process, in which case the application makes explicit calls to perform request processing, typically after each frame is drawn. The system is constructed to minimise the interference of the model manager with the application's

normal tasks, emphasising the freedom of the application to store data as it chooses. Since data delivery is a one-way, atomic process, there is no need for any blocking of the application in a multi-process implementation during normal scene graph traversal. Blocking can only occur when the application wishes to affect the structures inside the model manager through requests and priority alterations, both occurring only during the pre-fetch stage. The likelihood of conflicts in these regions is minimised since the model manager only requires very short term access to these structures when starting or completing a fetch operation, while most time is spent on the actual fetch operation itself.

As expected, application start-up latency is significantly reduced; a constant time instead of the time taken to load the whole model. Object fetching occurs during execution instead, and while the file access times remain small, and during periods of low stress, this is not very noticeable. During longer data transfers, the application process can be blocked from redrawing because the model manager process is using a large quota of CPU time for its fetching and processing. To alleviate this on a single processor machine requires finer control over process scheduling and input routines. Currently, no attempt is made to abort stalled or slow retrievals. Periodically checking that the priority for the segment being fetched is still high enough to merit the transfer continuing would address this. Similarly, segments, after being requested, might need removing from the request queue if their priorities drop significantly due to changing viewer demands.

As a demonstration of the lazy loading features, a model of the Sierpinski tetrahedron has been constructed in a manner such that displaying the detail within each tetrahedron involves forcibly loading a new data segment. From a hierarchy potentially infinite in size, the model manager successfully maintains a subset in memory for the application. A more realistic example is the terrain data (Figure 6.17) which is loaded as the viewer approaches the landscape. The central image shows a situation where rapid viewer movement has beaten the pre-fetch action and, in order to see more detail, the viewer has to wait for the data to be fetched, but all the while being able to move around a valid representation of the model.

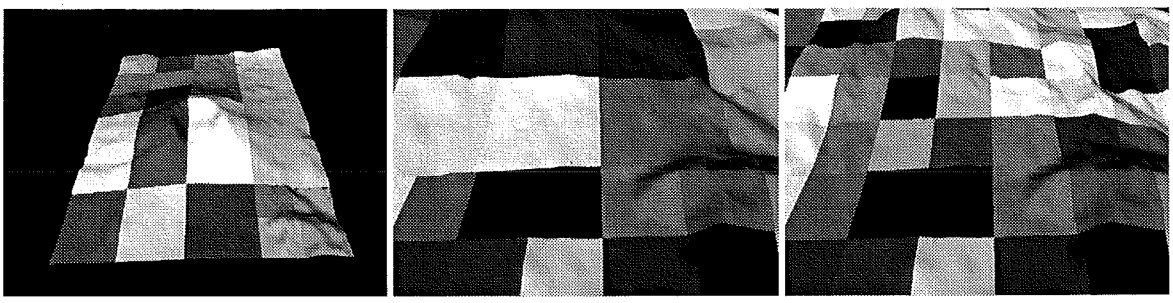


Figure 6.17: A terrain data example, false coloured to emphasize the different segments being loaded. The initial data loads quickly *(left)*, but after rapid viewer movement *(centre)* not all the higher-detail segments have been loaded and the viewer has to wait for them *(right)*.

As a final test for the system, a large model of a city was constructed through a random generation process including simple hierarchical approximations — it contains an estimated 500 million polygons in total and is far too large to load all at once. Figure 6.19 shows a series of views from a fly-past of the city model, zooming from an overview of the city down to the level of the cars, street lamps and trees. The corresponding graph of model manager activity (Figure 6.18) shows the fetch and removal activity as the manager keeps the number of segments loaded under a specified maximum. Figure 6.20 demonstrates the difference the quite low memory limit makes on images in which larger portions of the model are visible. The use of simple representations stored separately from the higher-quality ones becomes very important with distributed models where data retrieval times are quite large. Sensible segmentation also means that in many cases the more complex representation is never loaded since it is never needed.

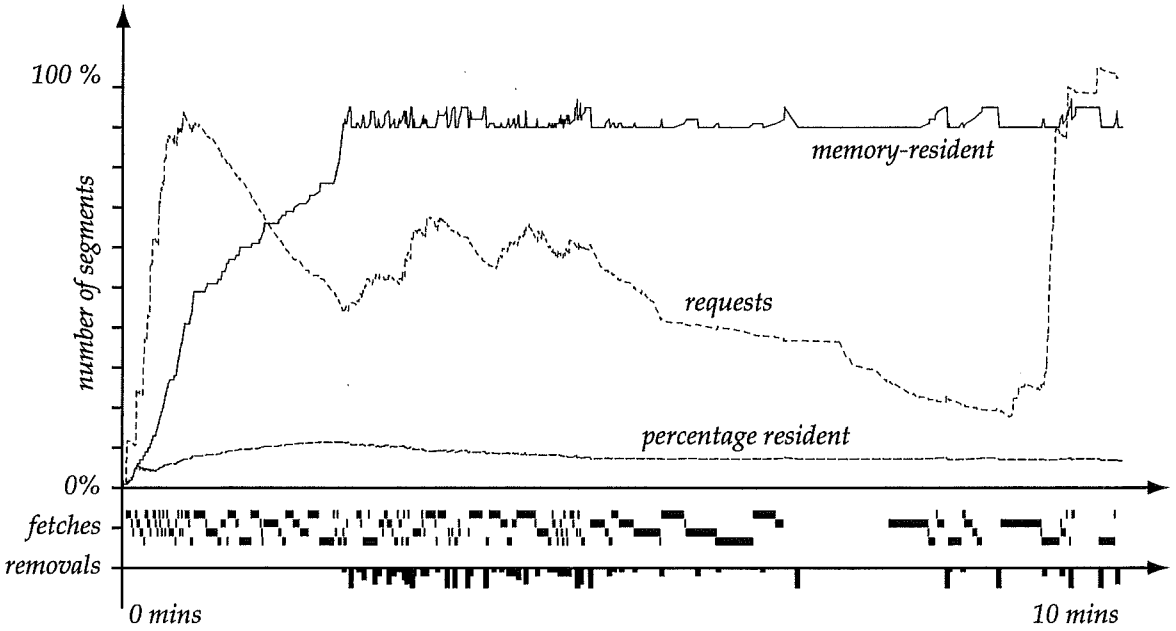


Figure 6.18: Fetch and removal activity during a ten minute exploration of the city model similar to the images in Figure 6.19. The upper graph shows the number of memory-resident segments, and also the number of requests queued, as time progresses.

Compared to the architectural walk-through system described in Section 6.2, the overheads are higher since the model is not stored externally in an internal format. This is balanced by the ability to utilise any data without clever pre-processing, although full advantage can be taken of data which is in a particularly suitable form. The functionality of the model manager system is quite similar to that of VRML browsers (*e.g.* [**Webspace**] [**Balaguer95**]) in that a distributed model is loaded during rendering. VRML browsers, however, are not selective about what to load and lack the capability to convert object types automatically. More

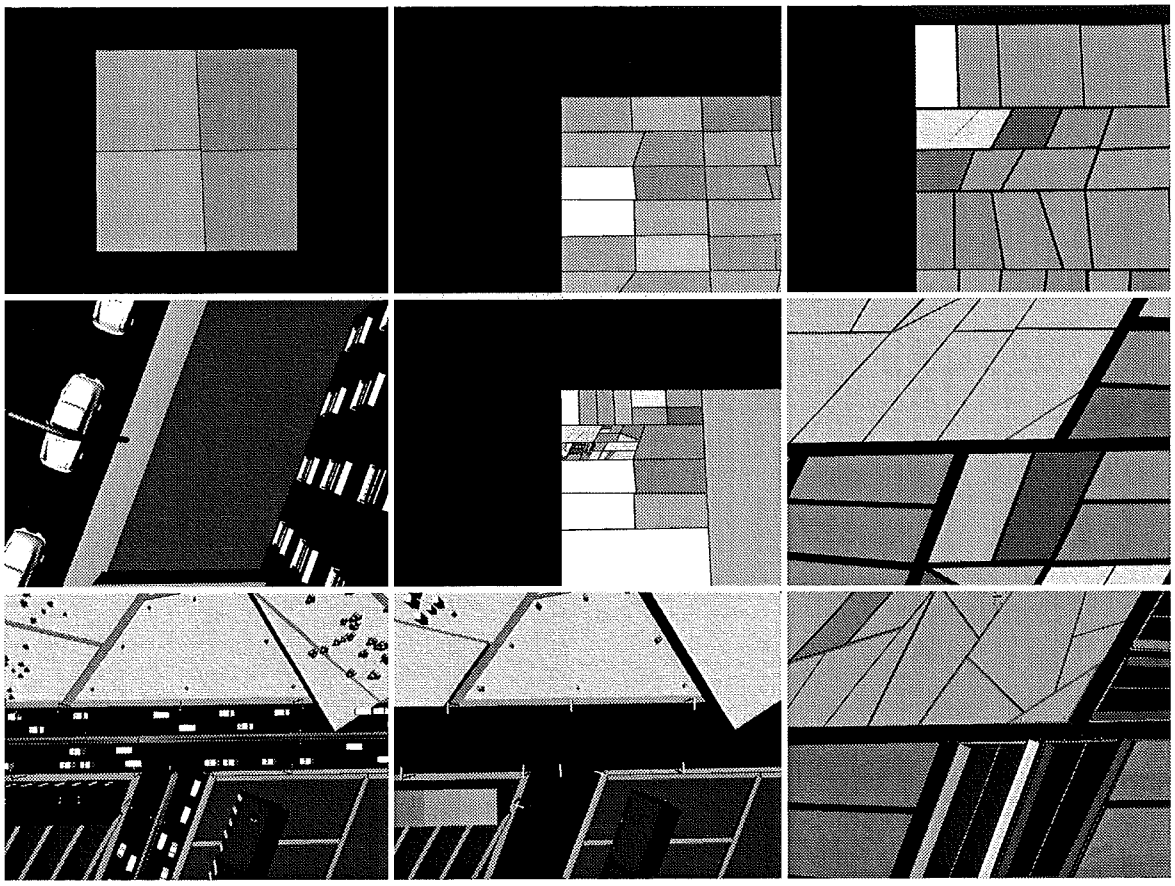


Figure 6.19: A sequence of eight images taken from an interactive exploration of the city model within a budget of 100 segments *(clockwise from top left)*. The centre image shows the entire model at high quality immediately after jumping out from the zoomed-in viewpoint, demonstrating the concentration of detail around the previous viewpoint.

importantly, they are not designed to work for an associated application. With the system described, the application needs only a minimal interaction with the model manager, and can otherwise assume that data appears on demand.

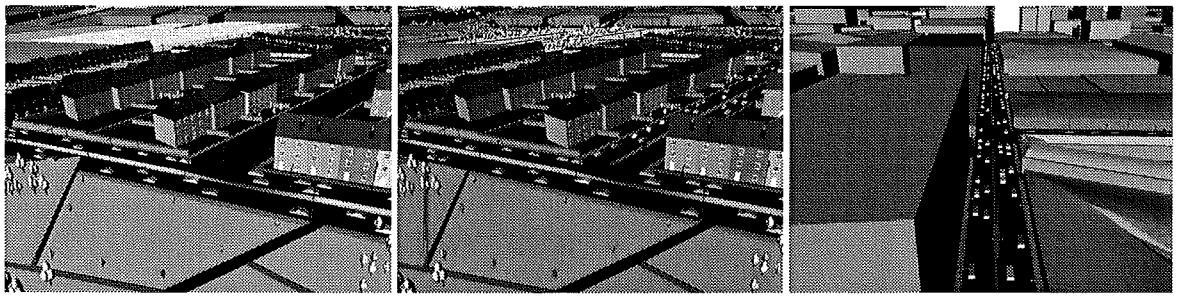


Figure 6.20: Three further images from the city model showing how less important detail is removed when constrained to 100 segments *(left)* compared to the same view with a much larger budget *(centre)*, and finally another view with a limit of 300 segments.

6.6 Summary

This chapter has described in detail how model managers can be constructed to encapsulate the tasks necessary for handling complex databases. A valid partial model is maintained within the resources available through the novel properties of approximation hierarchies. This incorporates the distributed model aspects of VRML and extends them to incorporate lazy loading and transparent format and type conversion. The implementation of an example model manager which successfully applies many of the mechanisms advocated in this dissertation has also been described. This implementation has demonstrated how it is possible to construct systems which can fulfil the aims set out in Chapter 1.

Chapter 7

Conclusions and Further Work

— In which the achievements of this work are assessed.

The previous chapters have described the key mechanisms necessary to construct systems that fulfil the aims set out in Chapter 1. This final chapter returns to these aims and examines the progress that has been made towards achieving them. This chapter also includes a summary of areas which would benefit from further research, and anticipates some of these research directions that are of particular interest.

7.1 Summary

In order to use arbitrarily complex models within a finite budget of resources it has been argued that it is necessary to be able to reduce complexity to a manageable level. When rendering a particular image, two complementary mechanisms are available. Visibility culling has been studied previously and concentrates on identifying objects that will definitely not be visible in the image and play no part in its calculation. This can still leave large amounts of visible detail that cannot be discarded. This dissertation has described how to use different quality representations of objects in order to reduce the potentially-visible complexity.

While this is already done in an ad hoc manner by creating different levels of detail for objects manually, this is not sufficient to meet future requirements. It is important to have algorithms that can produce different quality representations automatically for individual objects and also for groups of objects. The key to scalability is a hierarchical structure, allowing arbitrary levels of simplification. Approximation hierarchies have been defined

here as a flexible structure for achieving this.

While dedicated simplification routines will always have an important part to play, there is an immediate need for algorithms that can calculate replacements for *groups* of objects, in order to construct a hierarchy of approximations. A new practical approach has been described for analysing groups of objects and extracting important visual information which can be used as a basis for calculating replacement objects. Replacement schemes that generate oriented, possibly transparent, boxes and ellipsoids have been described that fulfil this need.

Multiple representations of objects can also be used as a means of hiding object types and formats from users. Together with the possibilities for using quality variation in a creative way, this can be used to aid the design process. In particular, the transfer and exchange of objects will be facilitated, resulting in a much wider use of objects and parts from standard libraries. Without such advances, the creation of complex models will place an unnecessarily heavy burden on designers.

Approximation should not be limited to geometrical data. Other aspects of models such as surface properties can also be included, extending from the geometry to light sources and eventually to rendering and processing algorithms. By using quality controls coherently throughout a system, an appropriate computational effort can be expended on the data being processed.

Once an approximation hierarchy is established and a visibility culling scheme employed, it is possible to free the model from being constrained inside the application, leading to a model-centred system. For a complex model, this finally removes the restrictions on the size of the model — it no longer has to be small enough to fit in the memory of any computer on which it is to be used. An application need load only the portion of the model that is required at any one time.

The final contribution of this work is the proposal for a software subsystem that can isolate many of these tasks from the application and user. The model manager presents a clean interface to the application, hiding as many aspects of the partial model as possible, and presenting it as if it was just a window on a virtual model. Furthermore, the application can be unaware of any conversions or approximations that might be necessary to provide the data in the form that it requires.

An example of each of these crucial mechanisms has been successfully implemented to demonstrate the feasibility of these ideas. The preliminary results of these experiments confirm that it is possible to achieve the aims for a system that can use very complex models.

7.2 Conclusions

As model complexity increases, the storage capacity and communications bandwidth required for managing the model will increase linearly. Most current rendering methods grow at best linearly with the size of the model, while ray-tracing and radiosity approaches are mainly between linear and quadratic (Figure 7.1).

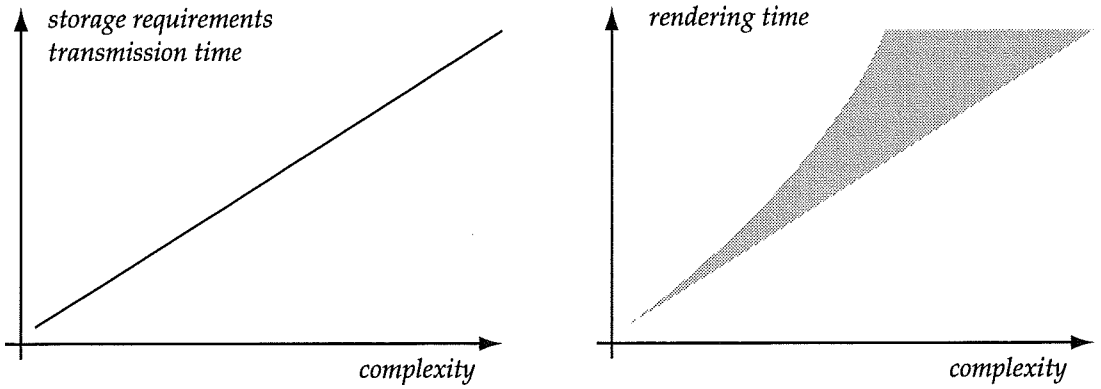


Figure 7.1: The current dependency on model complexity of storage and communication requirements and rendering time.

For the majority of images, this behaviour does not represent the complexity of the result. The aim of this work has been to reduce these curves to the sets shown in Figure 7.2. The additional quality parameter allows the user control over the quality of the results and enables the size of the computation to be matched to the physical resources and time available.

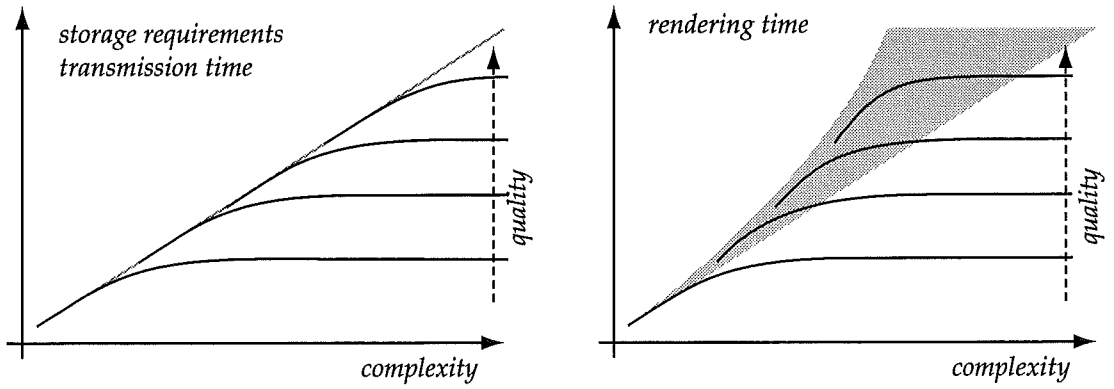


Figure 7.2: How rendering time and storage size need to depend on complexity in the future if scalable systems are to be achieved.

In terms of the aims put forward in Chapter 1, this dissertation has addressed the following requirements.

Scalable and Distributable The introduction of approximation hierarchies enables partial models to be extracted from any size of model. Furthermore, the model can be broken up into small segments which are dynamically retrieved only when needed.

General and Flexible No constraints are made on the types of objects present in an approximation hierarchy, and by providing conversion and approximation routines, objects of one type or format can be transformed for use in a system which does not explicitly support them.

Controllable and Consistent The use of quality as a control throughout a graphics system is very attractive and intuitive for users. Many algorithms are already controllable in terms of the quality of the results they generate, and these need to be incorporated into the general framework.

Portable Quality indirectly controls the demands made on resources in handling complex models, and therefore allows the applications on different configurations of hardware to access the same model data.

It is important that the techniques applied to the example components described in this dissertation are also applied consistently to other areas in order to construct scalable and portable systems that can cope with the demands placed on them by increased complexity.

7.3 Further Work

This dissertation has introduced a number of mechanisms that solve the problems associated with complex databases. There are, however, many areas described that require further work, and many that have the scope to incorporate new ideas. The following areas arise from the needs identified in the early chapters of this dissertation, and the mechanisms introduced to fulfil them.

Other Application Areas

Computer graphics is not solely concerned with synthesising 2D images from 3D models. There is a growing opinion that the "multi-resolution" approach should be built-in to all aspects of computer systems [**Read92**]. Certainly with increasing interest in multi-media and networking and the huge volumes of information that are becoming accessible, there is an urgent demand for data management and filtering. Increases in complexity will require more thought to be placed on user interface design in order to help users to obtain the important information.

For example, the majority of the results of this work can be used directly within 2D drafting and CAD systems which can suffer from most of the problems associated with 3D graph-

ics. Image processing is notoriously resource intensive, but also benefits from rapid feedback. Applying multi-resolution techniques enables rapid previewing but without sacrificing longer-term fidelity and control.

Approximation Algorithms

Automated approximation algorithms are central to the success of quality-based systems. Both rapid previews and high quality results are needed to enable all users to access complex models. The attribute gathering schemes developed in Chapter 3 need to be widened to capture more information about objects. These other attribute values could then be used to increase the range of the simple replacements proposed here. Approximations also need to be extended to all modelling paradigms including procedural and other higher-level modelling systems where generating good approximations should be easier. Efficient and useful means of measuring quality also need to be developed to quantify the approximations that are generated from these algorithms.

Quality Control

The description of quality, and the control of quality in human, rather than machine, terms continues to be an area for improvement. Since most images are destined for viewing through the human visual and perceptual system, more attention needs to be paid to how this can be utilised in designing graphics systems. Recent attempts to use image-based quality metrics include [**Maciel94**] and [**Sillion95**].

Model Creation

Model creation still remains the most significant bottleneck in the process of generating images of complex models. More advanced design tools, and an increasing amount of automation and computer-generated models are part of the solution. For example, physically-based and constraint-based modelling systems, and higher-level specifications together with the potential for automatic detail generation methods, offer cheap ways in which demands for complexity can be met. Encouraging the sharing of resources and the creation of distributed models is also a crucial means of assisting model designers.

Model Structuring and Retrieval

Having provided the means to only use a subset of the model at once, more attention needs to be paid to efficient methods for identifying *in advance* which parts of an approximation hierarchy are needed for the next image, and also which are likely to be needed soon. Visibility structuring and prediction is still a time-consuming and often primitive process, and

future systems will introduce more complex criteria defining their needs. With very complex models arguably the most important part of rendering will be the accurate prediction of which data is needed, since it can reduce the delay in fetching data, the memory to store it and the rendering time involved in creating the final image.

Progressive Rendering

One of the more pressing aims is to bridge the gulf between fast low-quality rendering algorithms and the slower high-quality ones. Approximation hierarchies provide the tools for describing the model at a whole range of qualities and there is an opportunity for rendering algorithms that use this data progression effectively. For example, a rendering system can be imagined that, instead of having to start again whenever the quality is changed, can simply proceed to increase the quality of the image displayed in much the same way an artist would first block out a painting and slowly refine the details.

This dissertation has not attempted to categorically answer, or even anticipate, all the problems that may arise, and one of the conclusions that has been drawn is that any such answers can only be short-term, and that continual review is always necessary. Many of the underlying themes of this work were first expressed in papers such as **[Clark76]** or **[Rubin80]** at a time when the complexity considered was of a different order of magnitude. The possibility of widely-available, interactive, high-quality image generation provides another incentive to examine the concepts of quality control and apply them to create flexible systems that can meet the demands of the future.

Appendix A

Surface Properties

A.1 Surface Inertia Tensors

This section contains descriptions of surface areas, centres of mass and surface inertia tensors for several common geometrical primitives. These can be transformed according to Equations 4.26, 4.25 and 4.27 to more general primitives.

A.1.1 Triangles

Since all surfaces can be decomposed into triangles, tensors for more compex surfaces, such as polygons, can be computed by linearly combining the results for single triangles.

For a triangle T with vertices at $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$, any internal point can be expressed as

$$\boldsymbol{x} = \boldsymbol{a} + u\boldsymbol{s} + v\boldsymbol{t} \tag{A.1}$$

where

$$\boldsymbol{s} = \boldsymbol{b} - \boldsymbol{a} \quad \text{and} \quad \boldsymbol{t} = \boldsymbol{c} - \boldsymbol{a} \tag{A.2}$$

so that the triangle corresponds to the region $D = \{\, v \in [0,1],\ u \in [0, 1-v] \,\}$ and has area

$$A_T = \frac{1}{2}|\boldsymbol{s} \times \boldsymbol{t}| \tag{A.3}$$

The surface properties of T are all computed by integrals of the form

$$I_f = \int_T f(\boldsymbol{x})\, dS \tag{A.4}$$

which can be transformed to integrals with respect to u and v

$$I_f = \int_D f(u,v)\left|\frac{\partial \boldsymbol{x}}{\partial u} \times \frac{\partial \boldsymbol{x}}{\partial v}\right| du\, dv \tag{A.5}$$

$$= |\boldsymbol{s} \times \boldsymbol{t}| \int_D f(u,v)\, du\, dv$$

$$= 2 \cdot A_T \int_D f(u,v)\, du\, dv \tag{A.6}$$

Using this, the six integrals needed to compute the inertia tensor are

$$\begin{aligned} I_1 &= \int_0^1 \int_0^{1-v} 1\, du\, dv = 1/2 \\ I_u = I_v &= \int_0^1 \int_0^{1-v} v\, du\, dv = 1/6 \\ I_{u^2} = I_{v^2} &= \int_0^1 \int_0^{1-v} v^2\, du\, dv = 1/12 \\ I_{uv} &= \int_0^1 \int_0^{1-v} uv\, du\, dv = 1/24 \end{aligned} \tag{A.7}$$

The inertia tensor is then combinations of terms of the form

$$\begin{aligned} \int_T x_i x_j\, dS &= a_i a_j I_1 + a_i s_j I_u + a_j s_i I_u + a_i t_j I_v + a_j t_i I_v \\ &\quad + s_i s_j I_{u^2} + t_i t_j I_{v^2} + s_i t_j I_{uv} + s_j t_i I_{uv} \end{aligned} \tag{A.8}$$

$$\begin{aligned} \int_T x_i^2\, dS &= a_i^2 I_1 + 2 a_i s_i I_u + 2 a_j t_i I_v \\ &\quad + s_i^2 I_{u^2} + t_i^2 I_{v^2} + 2 s_i t_i I_{uv} \end{aligned} \tag{A.9}$$

When simplified, this results in an inertia tensor which is equivalent to four point masses:

- three with $1/12$th A_T at each vertex $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$
- and one with $3/4$ A_T at its centre of mass, $(\boldsymbol{a} + \boldsymbol{b} + \boldsymbol{c})/3$.

A.1.2 Spheres

A sphere S of radius r centred at the origin has surface area

$$A_S = 4\pi r^2 \tag{A.10}$$

and its surface inertia tensor is

$$\boldsymbol{T}_S = A_S \cdot \begin{pmatrix} \frac{2}{3}r^2 & 0 & 0 \\ 0 & \frac{2}{3}r^2 & 0 \\ 0 & 0 & \frac{2}{3}r^2 \end{pmatrix} \tag{A.11}$$

A.1.3 Boxes

An axis-aligned box B with sides of length a, b and c centred at the origin has surface area

$$A_B = 2(ab + bc + ca) \tag{A.12}$$

and its surface inertia tensor can be expressed as

$$\boldsymbol{T}_B = A_B \cdot \begin{pmatrix} \frac{1}{6}(\beta b^2 + \gamma c^2) & 0 & 0 \\ 0 & \frac{1}{6}(\alpha a^2 + \gamma c^2) & 0 \\ 0 & 0 & \frac{1}{6}(\alpha a^2 + \beta b^2) \end{pmatrix} \tag{A.13}$$

where

$$\begin{array}{rcccl} \alpha & = & \frac{(ab+3bc+ca)}{(ab+bc+ca)} & = & 1 + \frac{2bc}{(ab+bc+ca)} \quad = \quad 1 + (4bc/A_B) \\ \beta & = & \frac{(ab+bc+3ca)}{(ab+bc+ca)} & = & 1 + \frac{2ca}{(ab+bc+ca)} \quad = \quad 1 + (4ac/A_B) \\ \gamma & = & \frac{(3ab+bc+ca)}{(ab+bc+ca)} & = & 1 + \frac{2ab}{(ab+bc+ca)} \quad = \quad 1 + (4ab/A_B) \end{array} \tag{A.14}$$

A.1.4 Cylinders and Discs

A closed cylinder C of length l and radius r with its axis along the z-axis and its centre at the origin has surface area

$$A_C = 2\pi r(r + l) \tag{A.15}$$

and surface inertia tensor

$$\boldsymbol{T}_C = A_C \cdot \begin{pmatrix} \tau/12(r + l) & 0 & 0 \\ 0 & \tau/12(r + l) & 0 \\ 0 & 0 & \frac{1}{2}r^2(r + 2l)/(r + l) \end{pmatrix} \tag{A.16}$$

where

$$\tau = 3r^3 + 6r^2 l + 3rl^2 + l^3 \tag{A.17}$$

A disc D of radius r with its normal along the z–axis centred at the origin has surface area

$$A_D = \pi r^2 \tag{A.18}$$

and surface inertia tensor

$$\boldsymbol{T}_D = A_D \cdot \begin{pmatrix} \frac{1}{4}r^2 & 0 & 0 \\ 0 & \frac{1}{4}r^2 & 0 \\ 0 & 0 & \frac{1}{2}r^2 \end{pmatrix} \tag{A.19}$$

A.2 Average Projected Surface Area Rule

This section gives a brief proof that the average projected surface area (*APSA*) of a convex closed object is one quarter of its total surface area. It is proved first for the case of a single unoccluded one-sided triangle and then extended to arbitrary closed convex surfaces. All projections P referred to are assumed to be orthogonal, and are thus described by a direction vector v only.

Triangle T is one-sided, so only makes a contribution when viewed from the front. It can be rotated without loss of generality so that it is in the yz-plane with one side along the z-axis and one corner at the origin. The triangle can the be described by three constants a, b, c so that it has corners at $(0,0,0)$, $(0,0,a)$ and $(0,b,c)$ (Figure A.1 *(left)*). Let n be the normal to the triangle, initially along the x-axis.

The projection P is fixed along the y-axis looking at the xz-plane — the triangle is rotated through all directions. The projected surface area will be calculated for the triangle with the normal vector rotated by ϕ around the y-axis, and then up from the xz-plane by an angle θ (Figure A.1 *(right)*). The front of the triangle can be seen along the projection while $0 \leq \theta \leq \pi$ and $-\pi / 2 \leq \phi \leq \pi / 2$. The projected area A_P is then given by

$$A_P = \begin{cases} \frac{1}{2} a \sin\phi \cos\theta & 0 \leq \theta \leq \pi,\ -\pi/2 \leq \phi \leq \pi/2 \\ 0 & \text{otherwise.} \end{cases} \tag{A.20}$$

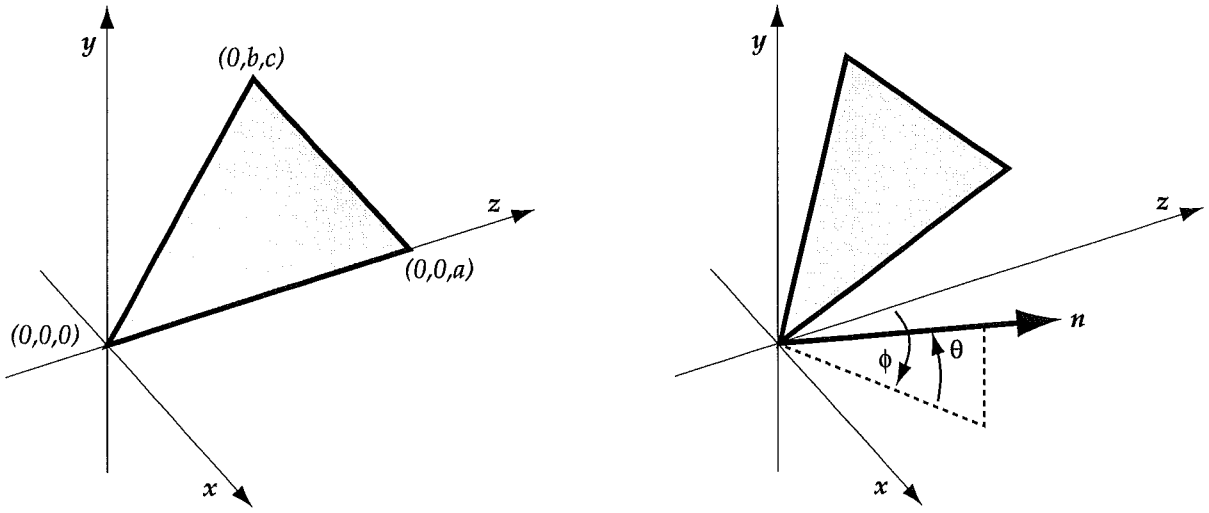


Figure A.1: The projection and coordinate systems used to describe the triangle when computing its average projected surface area.

To find the average projected surface area $\overline{A_P}$, the projected area A_P has to be integrated

over the range $0 \leq \theta \leq 2\pi$ to $-\pi/2 \leq \pi/2$ and then normalised:

$$\begin{aligned}
\overline{A_P} &= \int_0^{\pi} \int_{-\pi/2}^{\pi/2} \frac{1}{2} ac \sin\phi \cos\theta \cdot \cos\theta \, d\theta \, d\phi \Big/ \\
&\qquad \int_0^{2\pi} \int_{-\pi/2}^{\pi/2} \cos\theta \, d\theta \, d\phi \qquad \text{(A.21)} \\
&= \frac{1}{2} ac \int_0^{\pi} \sin\phi \, d\phi \int_{-\pi/2}^{\pi/2} \cos^2\theta \, d\theta \Big/ \int_0^{2\pi} d\phi \int_{-\pi/2}^{\pi/2} \cos\theta \, d\theta \\
&= \frac{1}{2} ac \cdot 2 \cdot \frac{\pi}{2} \Big/ 2\pi \cdot 2 \\
&= \frac{1}{2} ac \,/\, 4 \qquad \text{(A.22)}
\end{aligned}$$

which is one quarter of the triangle's surface area.

The average projected surface area for n non-occluding triangles is just the sum of the average projected surface areas of each of the triangles since the front of each triangle can never be occluded by another triangle if the shape is convex.

Since any closed convex surface can be constructed from triangles and when viewed only the front of these triangles is ever visible, the result holds for any closed, convex surface.

Appendix B

The Models

The colour images in this dissertation are all taken from applications built using the Open Inventor toolkit. Apart from a few standard datasets such as the teapot, the bust of Beethoven, the galleon, and the foot bones used in Chapter 4, and the terrain data from Mount St. Helens, all the models have been constructed for this work. These models fall into two categories — those generated by hand and those generated automatically by procedural methods. This section briefly describes two of the larger models.

The Office Models

Most of the objects in the office scenes (*e.g.* Figure 5.30 or Figure 5.29) were constructed manually from a selection of parts. These parts, such as all the fittings on the lamp, chair and seats, and even the leaves of the plant, are mainly "Generalised Cylinders", a restricted form of generative model that allows a constant cross-section to be swept along a curved path with varying radius and twist.

The piles of books are generated through a program which can generate random size, shape and colour books and either stack them on top of each other, or let them lean against the previous one on a shelf. Each book can be either a simple box approximation, a more complex cover plus interior, or a full book with a cover and an arbitrary number of pages.

The City Models

The city model used in Chapter 6 was generated randomly through a simple subdivision algorithm that recursively splits blocks into two sub-blocks. Splitting is performed so that roads have a high chance of coming out opposite one another across a join, leading to more natural street patterns. Constraints are imposed to try to generate usefully shaped areas, although in some cases the splitting algorithm could be improved upon. Each block can be of a number of types, such as residential, office blocks or parkland. Similarly there are a number of classes of roads which can split the blocks. Each block has a density and average height associated with it and this is used to determine the likely type of each sub-block after splitting occurs. At the lowest level, houses, trees, cars and so on are placed in the blocks and along the roads.

The 5km diameter model shown in Figure 6.20 occupies over 60Mbytes of disc space even with extensive use of shared instances and takes about two hours to generate. The process was designed to be random, but repeatable through the use of inherited random seeds, so that the models could be generated on need rather than in advance.

Bibliography

[Airey90] J. M. Airey, J. H. Rolf, and F. P. Brooks. **"Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments"**. *Computer Graphics*, volume 24 number 2, pages 41–50, 1990. (cited on page 147)

[Akeley93] K. Akeley. **"Reality Engine Graphics"**. In *Computer Graphics (SIGGRAPH '93 Conference Proceedings)*, pages 109–116, 1993. (cited on page 97)

[Amburn86] P. Amburn, E. Grant, and T. Whitted. **"Managing Geometric Complexity with Enhanced Procedural Models"**. In *Computer Graphics (SIGGRAPH '86 Conference Proceedings)*, pages 189–195, 1986. (cited on pages 5, 71)

[Atkinson83] M. P. Atkinson, P. J. Bailey, K. J. Chisholm, P. W. Cockshott, and R. Morrison. **"An Approach to Persistent Programming"**. *Computer Journal*, volume 26 number 4, pages 360–365, 1983. (cited on page 146)

[Aupperle93] L. Aupperle and P. Hanrahan. **"A Hierarchical Illumination Algorithm for Surfaces with Glossy Reflection"**. In *Computer Graphics (SIGGRAPH '93 Conference Proceedings)*, pages 155–162, 1993. (cited on page 131)

[Balaguer95] J-F. Balaguer and E. Gobetti. **"i3D: A High-Speed 3D Web Browser"**. Technical Report, Centre for Advanced Studies, Research and Development in Sardinia, 1995. (cited on page 164)

[Barr92] A. Barr. **"Rigid Physically Based SuperQuadrics"**. In *Graphics Gems III*, pages 137–159. 1992. (cited on page 92)

[Becker93] B. G. Becker and N. L. Max. **"Smooth Transitions between Bump Rendering Algorithms"**. In *Computer Graphics (SIGGRAPH '93 Conference Proceedings)*, pages 183–189, 1993. (cited on pages 98, 137)

[Bell95] A. G. Bell. **"VRML Basics"**. In *VRML: Using 3D to Surf the Web*, (Ed. J. Hardenbergh) *SIGGRAPH '95 Course Notes*, number 12. 1995. (cited on pages 150, 151)

[Bergman86] L. Bergman, H. Fuchs, and E. Grant. **"Image Rendering by Adaptive Refinement"**. In *Computer Graphics (SIGGRAPH '86 Conference Proceedings)*, pages 29–39, 1986. (cited on page 133)

[Biederman85] I. Biederman. **"Human Image Understanding"**. *Computer Vision, Graphics & Image Processing*, volume 32, pages 29–73, 1985. (cited on pages 74, 75, 83)

[Bishop94] G. Bishop, H. Fuchs, L. McMillan, and E. J. Scher Zagier. **"Frameless Rendering: Double Buffering Considered Harmful"**. In *Computer Graphics (SIGGRAPH '94 Conference Proceedings)*, pages 175–176, 1994. (cited on page 134)

[Blinn82] J. F. Blinn. **"A Generalization of Algebraic Surface Drawing"**. *ACM Transactions on Graphics*, volume 1 number 3, pages 235–256, July 1982. (cited on page 70)

[Borrel95] P. Borrel, K. Cheng, P. Darmon, P. Kirchner, J. Lipscomb, J. Menon, J. Mittleman, J. Rossignac, B. Schneider, and B. Wolfe. **"The IBM 3D Interaction Accelerator (3DIX)"**. Technical Report, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, 1995. (cited on page 57)

[Brechner95] E. Brechner. **"Future Directions"**. In *Interactive Walkthrough of Large Geometric Databases*, (Ed. E. Brechner) *SIGGRAPH '95 Course Notes*, number 33. 1995. (cited on pages 86, 94)

[Broll95] W. Broll and D. England. **"Bringing Worlds Together: Adding Multi-User Support to VRML"**. In *Proceedings of the VRML '95 Symposium*. ACM SIGGRAPH, December 1995. (cited on page 151)

[Brown95] P. J. C. Brown. **"Multiresolution Terrain Modelling (Research Proposal)"**. Technical Report, University Of Cambridge Computer Laboratory, 1995. (cited on page 62)

[Cabral87] B. Cabral, N. Max, and R. Springmeyer. **"Bidirectional Reflection Functions from Surface Bump Maps"**. In *Computer Graphics (SIGGRAPH '87 Conference Proceedings)*, pages 273–281, 1987. (cited on page 98)

[Castle95] O. M. Castle. **"Synthetic Image Generation for a Multiple-View Autostereo Display"**. PhD thesis, University of Cambridge, Computer Laboratory, 1995. (cited on page 115)

[Cazals95] F. Cazals, G. Drettakis, and C. Puech. **"Filtering, Clustering and Hierarchy Construction: a New Solution for Ray-Tracing Complex Scenes"**. *Eurographics 1995 Conference Proceedings (Computer Graphics*

Forum), volume 14 number 3, pages 371–383, 1995. (cited on page 109)

[Charney90] M. J. Charney and I. D. Scherson. **"Efficient Traversal of Well-Behaved Hierarchical Trees of Extents for Ray-Tracing Complex Scenes"**. *Visual Computer*, volume 6 number 3, pages 167–178, June 1990. (cited on pages 26, 27)

[Chase86] W. G. Chase. **"Visual Information Processing"**, volume II of *Handbook of Perception and Human Performance*, chapter 35, (Eds. Bofl, Kaufman, and Thomas). Wiley, 1986. (cited on pages 50, 107)

[Chen91] S. E. Chen, H. E. Rushmeier, G. Miller, and D. Turner. **"A Progressive Multi-Pass Method for Global Illumination"**. In *Computer Graphics (SIGGRAPH '91 Conference Proceedings)*, pages 165–174, 1991. (cited on page 133)

[Chen95] S. E. Chen. **"QuickTime VR - An Image-Based Approach to Virtual Environment Navigation"**. In *Computer Graphics (SIGGRAPH '95 Conference Proceedings)*, pages 29–38, 1995. (cited on page 104)

[Chin89] N. Chin. **"Near Real-Time Shadow Generation Using BSP Trees"**. In *Computer Graphics (SIGGRAPH '89 Conference Proceedings)*, pages 99–105, 1989. (cited on page 130)

[Chiu94] K. Chiu and P. Shirley. **"Rendering, Complexity, and Perception"**. In *Proceedings of the 5th Eurographics Rendering Workshop*, 1994. (cited on page 132)

[Cignoni94] P. Cignoni, C. Montani, and R. Scopigno. **"MagicSphere: an Insight Tool for 3D Data Visualisation"**. *Eurographics 1994 Conference Proceedings (Computer Graphics Forum)*, volume 13 number 3, pages 317–328, 1994. (cited on pages 9, 139)

[Clark76] J. H. Clark. **"Hierarchical Geometric Models for Visible Surface Algorithms"**. *Communications of the ACM*, volume 19 number 10, pages 547–554, October 1976. (cited on pages 37, 39, 172)

[Cohen88] M. F. Cohen, S. E. Chen, J. R. Wallace, and D. P. Greenberg. **"A Progressive Refinement Approach to Fast Radiosity Image Generation"**. In *Computer Graphics (SIGGRAPH '88 Conference Proceedings)*, pages 75–84, 1988. (cited on page 133)

[Cohen94] M. Cohen and J. Wallace. **"Radiosity and Realistic Image Synthesis"**. Academic Press, 1994. (cited on page 131)

[Cook82] R. L. Cook and K. E. Torrance. **"A Reflectance Model for Computer Graphics"**. *ACM Transactions on Graphics*, volume 1 number 1, pages 7–24, January 1982. (cited on page 97)

[Cook84] R. L. Cook. **"Shade Trees"**. In *Computer Graphics (SIGGRAPH '84 Conference Proceedings)*, pages 223–231, 1984. (cited on page 98)

[Cosman90] M. A. Cosman, A. E. Mathisen, and J. A. Robinson. **"A New Visual System to Support Advanced Requirements"**. In *IMAGE V Conference*, 1990. (cited on page 138)

[Crow82] F. C. Crow. **"A More Flexible Image Generation Environment"**. In *Computer Graphics (SIGGRAPH '82 Conference Proceedings)*, pages 9–17, 1982. (cited on page 37)

[Deering95] M. Deering. **"Geometry Compression"**. In *Computer Graphics (SIGGRAPH '95 Conference Proceedings)*, pages 13–20, 1995. (cited on page 11)

[DeHaemer91] M. J. DeHaemer and M. J. Zyda. **"Simplification of Objects Rendered by Polygonal Approximations"**. *Computers & Graphics*, volume 15 number 2, pages 175–184, 1991. (cited on page 55)

[Dixon89] G. N. Dixon, G. D. Parrington, S. K. Shrivastava, and S. M. Wheater. **"The Treatment of Persistent Objects in Arjuna"**. Technical Report TR 283, University of Newcastle upon Tyne, 1989. (cited on page 146)

[Dorsey91] J. O'B. Dorsey, F. Sillion, and D. P. Greenberg. **"Design and Simulation of Opera Lighting and Projection Effects"**. In *Computer Graphics (SIGGRAPH '91 Conference Proceedings)*, pages 41–50, 1991. (cited on page 129)

[Duff92] T. Duff. **"Interval Arithmetic and Recursive Subdivision for Implicit Functions and Constructive Solid Geometry"**. In *Computer Graphics (SIGGRAPH '92 Conference Proceedings)*, pages 131–138, 1992. (cited on page 66)

[Eck95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsberyy, and W. Stuetzle. **"Multiresolution Analysis of Arbitrary Meshes"**. Technical Report TR #95-01-02, University of Washington, Seattle, January 1995. (cited on page 59)

[Erikson95] C. Erikson. **"Error Correction of a Large Architectural Model: The Henderson County Courthouse"**. Technical Report TR95-013, University of North Carolina, Chapel Hill, 1995. (cited on page 7)

[Falby93] J. S. Falby, M. J. Zyda, D. R. Pratt, and R. L. Mackey. **"NPSNET: Hierarchical Data Structures for Real-time Three-dimensional Visual Simulation"**. *Computers & Graphics*, volume 17 number 1, pages 65–69, 1993. (cited on pages 21, 55, 147)

[Floriani92] L. De Floriani and E. Puppo. **"A Hierarchical Triangle-Based Model for Terrain Description"**. In *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, (Eds. U. Formentini, A. U. Frank, and I. Campari), pages 236–251. Springer-Verlag, Lecture Notes in Computer Science, N.639, 1992. (cited on pages 52, 62)

[Foley90] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. **"Computer Graphics, Principles and Practice, Second Edition"**. Addison-Wesley, Reading, Massachusetts, 1990. (cited on pages 10, 20, 32, 63, 66, 67, 95, 96, 147)

[Forsey91] D. R. Forsey. **"Hierarchical Free-Form Surface Modelling"**. In *Topics in the Construction, Manipulation and Assessment of Spline Surfaces*, (Ed. Bartels) *SIGGRAPH '91 Course Notes*, number 25. 1991. (cited on page 64)

[Fournier94] A. Fournier. **"Introduction to Wavelets"**. In *Wavelets and their Applications in Computer Graphics*, (Ed. A. Fournier) *SIGGRAPH '94 Course Notes*, number 11. 1994. (cited on page 68)

[Fuchs80] H. Fuchs. **"On Visible Surface Generation by A Priori Tree Structures"**. In *Computer Graphics (SIGGRAPH '80 Conference Proceedings)*, pages 124–133, 1980. (cited on page 25)

[Fujimoto86] A. Fujimoto, T. Tanake, and K. Iwata. **"ARTS: Accelerated Ray-Tracing System"**. *IEEE Computer Graphics & Applications*, volume 6 number 4, pages 16–26, April 1986. (cited on pages 25, 27)

[Funkhouser92] T. A. Funkhouser, C. H. Sequin, and S. J. Teller. **"Management of Large Amounts of Data in Interactive Building Walkthroughs"**. *ACM SIGGRAPH Symposium on Interactive Computer Graphics*, pages 11–19, 1992. (cited on pages 10, 147, 158)

[Funkhouser93] T. A. Funkhouser and C. H. Sequin. **"Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments"**. In *Computer Graphics (SIGGRAPH '93 Conference Proceedings)*, pages 247–254, 1993. (cited on pages 28, 135, 136)

[Funkhouser95] T. Funkhouser. **"Database Management"**. In *Interactive Walkthrough of Large Geometric Databases*, (Ed. E. Brechner) *SIGGRAPH '95 Course Notes*, number 33. 1995. (cited on page 150)

[Glassner84] A. S. Glassner. **"Space Subdivision for Fast Ray-Tracing"**. *IEEE Computer Graphics & Applications*, volume 4 number 10, pages 15–22, October 1984. (cited on page 25)

[Glassner88] A. S. Glassner. **"Spacetime Ray Tracing for Animation"**. *IEEE Computer Graphics & Applications*, volume 8 number 2, pages 60–70, March 1988. (cited on page 108)

[Glassner93] A. S. Glassner. **"An Introduction to Ray Tracing"**. Academic Press, 1993. (cited on pages 20, 22, 122)

[Goldsmith87] J. Goldsmith and J. Salmon. **"Automatic Creation of Object Hierarchies for Ray Tracing"**. *IEEE Computer Graphics & Applications*, volume 7 number 5, pages 14–20, May 1987. (cited on pages 26, 80, 108, 109)

[Goldstein50] H. Goldstein. **"Classical Mechanics"**. Addison Wesley, 1950. (cited on page 83)

[Goldstein86] E. B. Goldstein. **"Sensation and Perception"**. Belmont, California, 1986. (cited on pages 106, 107)

[Gondek94] J. S. Gondek, G. W. Meyer, and J. G. Newman. **"Wavelength Dependent Reflectance Functions"**. In *Computer Graphics (SIGGRAPH '94 Conference Proceedings)*, pages 213–220, 1994. (cited on page 98)

[Goral84] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Bataille. **"Modelling the Interaction of Light Between Diffuse Surfaces"**. In *Computer Graphics (SIGGRAPH '84 Conference Proceedings)*, pages 213–222, 1984. (cited on page 131)

[Gortler93] S. J. Gortler, P. Schroder, M. F. Cohen, and P. Hanrahan. **"Wavelet Radiosity"**. In *Computer Graphics (SIGGRAPH '93 Conference Proceedings)*, pages 221–230, 1993. (cited on page 132)

[Greenberg91] D. P. Greenberg. **"More Accurate Simulations at Faster Rates"**. *IEEE Computer Graphics & Applications*, volume 11 number 1, pages 23–29, January 1991. (cited on page 13)

[Greene86] N. Greene. **"Environment Mapping and Other Applications of World Projections"**. *IEEE Computer Graphics & Applications*, volume 6 number 11, pages 21–29, November 1986. (cited on page 105)

[Greene93] N. Greene, M. Kass, and G. Miller. **"Hierarchical Z-Buffer Visibility"**. In *Computer Graphics (SIGGRAPH '93 Conference Proceedings)*, pages 231–238, 1993. (cited on page 27)

[Greene94] N. Greene and M. Kass. **"Error-Bounded Antialiased Rendering of Complex Environments"**. In *Computer Graphics (SIGGRAPH '94 Conference Proceedings)*, pages 59–66, 1994. (cited on page 27)

[Haerbili90] P. Haerbili. **"Paint By Numbers: Abstract Image Representations"**. In *Computer Graphics (SIGGRAPH '90 Conference Proceedings)*, pages 207–214, 1990. (cited on page 8)

[Hall91] R. Hall, M. Bussan, P. Georgiades, and D. P. Greenberg. **"A Testbed for Architectural Modelling"**. In *Eurographics '91*, pages 47–58. North-Holland, 1991. (cited on page 3)

[Hanrahan86] P. Hanrahan. **"Using Caching and Breadth-First Search to Speed up Ray-Tracing"**. In *Proceedings of Graphics Interface '86*, pages 56–61. Canadian Information Processing Society, 1986. (cited on page 134)

[Hanrahan91] P. Hanrahan, D. Salzman, and L. Aupperle. **"A Rapid Hierarchical Radiosity Algorithm"**. In *Computer Graphics (SIGGRAPH '91 Conference Proceedings)*, pages 197–206, 1991. (cited on pages 32, 131)

[Hart91] J. C. Hart and T. A. DeFanti. **"Efficient Antialiased Rendering of 3-D Linear Fractals"**. In *Computer Graphics (SIGGRAPH '91 Conference Proceedings)*, pages 91–100, 1991. (cited on pages 29, 118)

[Heckbert84] P. S. Heckbert and P. Hanrahan. **"Beam Tracing Polygonal Objects"**. In *Computer Graphics (SIGGRAPH '84 Conference Proceedings)*, pages 119–127, 1984. (cited on page 134)

[Heckbert87] P. S. Heckbert. **"Ten Unsolved Problems in Rendering"**. *Graphics Interface '87, Workshop on Rendering Algorithms*, 1987. (cited on pages 3, 44)

[Heckbert94] P. S. Heckbert and M. Garland. **"Multiresolution Modeling for Fast Rendering"**. In *Proceedings of Graphics Interface '94*, pages 43–50. Canadian Information Processing Society, 1994. (cited on page 12)

[Heeger95] D. J. Heeger and J. R. Bergen. **"Pyramid-Based Texture Analysis/Synthesis"**. In *Computer Graphics (SIGGRAPH '95 Conference Proceedings)*, pages 229–238, 1995. (cited on page 98)

[Heisserman94] J. Heisserman. **"Generative Geometric Design"**. *IEEE Computer Graphics & Applications*, volume 14 number 2, pages 37–45, March 1994. (cited on page 5)

[Hinker93] P. Hinker and C. Hansen. **"Geometric Optimization"**. In *Visualization '93*, 1993. (cited on page 55)

[Hoppe93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. **"Mesh Optimization"**. In *Computer Graphics (SIGGRAPH '93 Conference Proceedings)*, pages 19–25, 1993. (cited on page 56)

[HTML] **"HTML 2.0 Proposed Standard Materials"**. WWW site "http://w3.org/pub/WWW/MarkUp/html-spec". (cited on page 150)

[Hubbold93] R. Hubbold, A. Murta, A. West, and T. Howard. **"Design Issues for Virtual Reality Systems"**. Technical Report, Advanced Interfaces Group, University of Manchester, U.K., 1993. (cited on page 9)

[Julesz81] B. Julesz. **"Textons, the Elements of Texture Perception, and their Interaction"**. *Nature*, volume 290, pages 91–97, March 1981. (cited on page 50)

[Kajiya83] J. T. Kajiya. **"New Techniques for Ray Tracing Procedurally Defined Objects"**. In *Computer Graphics (SIGGRAPH '83 Conference Proceedings)*, pages 91–99, 1983. (cited on page 29)

[Kajiya85a] J. T. Kajiya. **"Anisotropic Reflection Models"**. In *Computer Graphics (SIGGRAPH '85 Conference Proceedings)*, pages 15–21, 1985. (cited on page 124)

[Kajiya85b] J. T. Kajiya. **"The Rendering Equation"**. In *Computer Graphics (SIGGRAPH '85 Conference Proceedings)*, pages 143–150, 1985. (cited on page 95)

[Kay86] T. L. Kay and J. T. Kajiya. **"Ray Tracing Complex Scenes"**. In *Computer Graphics (SIGGRAPH '86 Conference Proceedings)*, pages 269–277, 1986. (cited on pages 24, 94)

[Kernighan78] B. W. Kernighan and D. M. Ritchie. **"The C Programming Language"**. Prentice-Hall, 1978. (cited on page 115)

[Kirk91] D. Kirk and J. Arvo. **"Unbiased Sampling Techniques for Image Synthesis"**. In *Computer Graphics (SIGGRAPH '91 Conference Proceedings)*, pages 153–156, 1991. (cited on page 125)

[Kolb91] C. E. Kolb. **"Rayshade User's Guide and Reference Manual"**, 1991. (cited on page 4)

[Lansdown95] J. Lansdown and S. Schofield. **"Expressive Rendering: A Review of Nonphotorealistic Techniques"**. *IEEE Computer Graphics & Applications*, volume 15 number 3, pages 29–37, May 1995. (cited on page 8)

[Leclerc94] Y. G. Leclerc and S. Q. Lau. **"TerraVision: A Terrain Visualisation System"**. Technical Report Note #540, SRI International, Menlo Park, California, 1994. (cited on page 12)

[Lindstrom95] P. Lindstrom, D. Koller, L. F. Hodges, W. Ribarsky, N. Faust, and G. Turner. **"Level-of-Detail Management for Real-Time Rendering of Phototextured Terrain"**. Technical Report, Visualization & Usability Center, Georgia Institute of Technology, 1995. (cited on page 52)

[Lorensen87] W. E. Lorensen and H. E. Cline. **"Marching Cubes: A High Resolution Surface Construction Algorithm"**. In *Computer Graphics (SIGGRAPH '87 Conference Proceedings)*, pages 163–169, 1987. (cited on pages 35, 70)

[Lounsberry94] M. Lounsberry. **"Multiresolution Analysis for Surfaces of Arbitrary Toplogical Type"**. PhD thesis, University of Washington, Seattle, 1994. (cited on pages 58, 65, 98)

[Luebke96] D. Luebke and C. Georges. **"Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets"**. *ACM SIGGRAPH Symposium on Interactive 3D Graphics*, 1996. (cited on page 28)

[MacDonald90] J. D. MacDonald and K. S. Booth. **"Heuristics for Ray Tracing Using Space Subdivision"**. *The Visual Computer*, volume 6 number 3, pages 153–166, June 1990. (cited on page 108)

[Maciel94] P. W. C. Maciel. **"Interactive Rendering of Complex 3D Models in Pipelined Graphics Architectures"**. Technical Report #403, Indiana University, Bloomington, May 1994. (cited on pages 37, 171)

[Maciel95a] P. W. C. Maciel. **"Visual Navigation of Large Environments using Textured Clusters"**. PhD thesis, Indiana University, Bloomington, 1995. (cited on pages 113, 136)

[Maciel95b] P. W. C. Maciel and P. Shirley. **"Visual Navigation of Large Environments using Textured Clusters"**. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics*, pages 95–102, 1995. (cited on pages 97, 100, 102, 108)

[Mallat89] S. G. Mallat. **"A Theory for Multiresolution Signal Decomposition: The Wavelet Representation"**. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, volume 11 number 7, pages 674–693, July 1989. (cited on page 50)

[Marr82] D. K. Marr. **"Vision: A Computational Investigation into the Human Representation and Processing of Visual Information"**. Freeman, San Francisco, 1982. (cited on page 107)

[McMillan95] L. McMillan and G. Bishop. **"Plenoptic Modelling: An Image-Based Rendering System"**. In *Computer Graphics (SIGGRAPH '95 Conference Proceedings)*, pages 38–46, 1995. (cited on page 105)

[Milne26] A. A. Milne. **"Winnie the Pooh"**. Methuen, 1926.

[Moore96] J. R. Moore, N. A. Dodgson, A. R. L. Travis, and S. R. Lang. **"Time-multiplexed color autostereoscopic display"**. In *Proceedings SPIE 2653, Stereoscopic Displays and Applications VII*, 1996. (cited on page 115)

[Mueller95] C. Mueller. **"Architectures of Image Generators for Flight Simulators"**. Technical Report, University of North Carolina, Chapel Hill, 1995. (cited on pages 21, 138, 147)

[Muraki91] S. Muraki. **"Volumetric Shape Description of Range Data using "Blobby Model""**. In *Computer Graphics (SIGGRAPH '91 Conference Proceedings)*, pages 227–235, 1991. (cited on page 70)

[Muraki93] S. Muraki. **"Volume Data and Wavelet Transforms"**. *IEEE Computer Graphics & Applications*, volume 13 number 4, pages 50–56, July 1993. (cited on page 68)

[Nishita86] T. Nishita and E. Nakamae. **"Continuous Tome Representation of Three-Dimensional Objects Illuminated by Skylight"**. In *Computer Graphics (SIGGRAPH '86 Conference Proceedings)*, pages 125–132, 1986. (cited on page 129)

[Nishita94] T. Nishita and E. Nakamae. **"A Method for Displaying Metaballs by using Bezier Clipping"**. *Eurographics 1994 Conference Proceedings (Computer Graphics Forum)*, volume 13 number 3, pages 271, 1994. (cited on page 70)

[OpenGL93] OpenGL Architecture Review Board. **"OpenGL Programming Guide"**, 1993. (cited on pages 16, 120)

[Painter89] J. Painter and K. Sloan. **"Antialiased Ray Tracing by Adaptive Progressive Refinement"**. In *Computer Graphics (SIGGRAPH '89 Conference Proceedings)*, pages 281–288, 1989. (cited on page 134)

[Pattanaik93] S. N. Pattanaik and S. P. Mudur. **"The Potential Equation and Importance in Illumination Computations"**. *Computer Graphics Forum*, volume 12 number 2, pages 131–136, 1993. (cited on page 131)

[Phong75] B. T. Phong. **"Illumination for Computer Generated Pictures"**. *Communications of the ACM*, volume 18 number 6, pages 311–317, 1975. (cited on page 96)

[Press92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. **"Numerical Recipes in C"**. Cambridge University Press, 2nd edition, 1992. (cited on page 84)

[Prusinkiewicz88] P. Prusinkiewicz, A. Lindenmayer, and J. Hanan. **"Developmental Models of Herbaceous Plants for Computer Imagery Purposes"**. In *Computer Graphics (SIGGRAPH '88 Conference Proceedings)*, pages 141–150, 1988. (cited on page 5)

[Puech90] C. Puech, F. Sillion, and C. Vedel. **"Improving Interaction with Radiosity-Based Lighting Simulation Programs"**. *Computer Graphics*, volume 24 number 2, pages 51–57, 1990. (cited on page 133)

[Quarendon93] P. Quarendon. **"Towards Three-Dimensional Models of Reality"**. In *Virtual Reality Systems*, (Eds. R. A. Earnshaw, N. A. Gigante, and H. Jones), pages 249–261. Academic Press, 1993. (cited on page 3)

[Read92] R. L. Read, D. S. Fussell, and A. Silberschatz. **"A Multi-Resolution Relational Data Model"**. Technical Report TR-92-10, University of Texas at Austin, 1992. (cited on pages 147, 170)

[Read93] R. L. Read, D. S. Fussell, and A. Silberschatz. **"System-Wide Multiresolution"**. Technical Report TR-93-04, University of Texas at Austin, 1993. (cited on pages 12, 15, 39)

[Reddy94] M. Reddy. **"A Motion-Sensitive Approach to Combating Inherent Latencies in VR Systems"**. Technical Report, University of Edinburgh, Computer Science Department, 1994. (cited on pages 118, 137)

[Reeves83] W. T. Reeves. **"Particle Systems – A Technique for Modeling a Class of Fuzzy Objects"**. In *Computer Graphics (SIGGRAPH '83 Conference Proceedings)*, pages 359–376, 1983. (cited on page 71)

[Reeves87] W. T. Reeves, D. H. Salesin, and R. L. Cook. **"Rendering Antialiased Shadows with Depth Maps"**. In *Computer Graphics (SIGGRAPH '87 Conference Proceedings)*, pages 283–291, 1987. (cited on page 130)

[Regan94] M. Regan and R. Pose. **"Priority Rendering with a Virtual Reality Address Recalculation Pipeline"**. In *Computer Graphics (SIGGRAPH '94 Conference Proceedings)*, pages 155–162, 1994. (cited on page 105)

[Rock86] I. Rock. **"Description and Analysis of Object and Event Perception"**, volume II of *Handbook of Perception and Human Performance*, chapter 33, (Ed. Thomas Boff, Kaufman). Wiley, 1986. (cited on page 75)

[Rohlf94] J. Rohlf and J. Helman. **"IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics"**. In *Computer Graphics (SIGGRAPH '94 Conference Proceedings)*, pages 381–394, 1994. (cited on pages 31, 102, 117, 147)

[Rossignac92] J. P. Rossignac and P. Borel. **"Multi-Resolution 3D Approximations for Rendering Complex Scenes"**. Technical Report, IBM Yorktown Heights, NY, 1992. (cited on page 57)

[Rubin80] S. Rubin and T. Whitted. **"A Three-Dimensional Representation for Fast Rendering of Complex Scenes"**. In *Computer Graphics (SIGGRAPH '80 Conference Proceedings)*, pages 110–116, 1980. (cited on pages 24, 172)

[Rushmeier93] H. Rushmeier, C. Patterson, and A. Veerasamy. **"Geometric Simplification for Indirect Illumination Calculations"**. *Proceedings of Graphics Interface '93*, pages 227–236, 1993. (cited on pages 94, 132)

[Saito90] T. Saito and T. Takahashi. **"Comprehensible Rendering of 3-D Shapes"**. In *Computer Graphics (SIGGRAPH '90 Conference Proceedings)*, pages 207–214, 1990. (cited on page 9)

[Salisbury94] M. P. Salisbury, S. E. Anderson, R. Barzel, and D. H. Salesin. **"Interactive Pen-and-Ink Illustration"**. In *Computer Graphics (SIGGRAPH '94 Conference Proceedings)*, pages 101–108, 1994. (cited on page 8)

[Samet90] H. Samet. **"Design and Analysis of Spatial Data Structures"**. Addison-Wesley, Reading, MA, 1990. (cited on pages 68, 108)

[Scarlatos90] L. L. Scarlatos. **"A Refined Triangulation Hierarchy for Multiple Levels of Terrain Detail"**. In *IMAGE V Conference 1990*, pages 115–123, 1990. (cited on page 52)

[Schröder95] P. Schröder and W. Sweldens. **"Spherical Wavelets: Efficiently Representing Functions on the Sphere"**. In *Computer Graphics (SIGGRAPH '95 Conference Proceedings)*, pages 161–172, 1995. (cited on pages 98, 130)

[Schroeder92] W. J. Schroeder, J. A. Zarge, and W. E. Lorenson. **"Decimation of Triangle Meshes"**. In *Computer Graphics (SIGGRAPH '92 Conference Proceedings)*, pages 65–70, 1992. (cited on page 55)

[Segal92] M. Segal, C. Korobkin, R. von Widenfelt, J. Foran, and P. Haeberli. **"Fast Shadows and Lighting Effects Using Texture Mapping"**. In *Computer Graphics (SIGGRAPH '92 Conference Proceedings)*, pages 249–252, 1992. (cited on page 130)

[Selgmann91] D. D. Selgmann and S. Feiner. **"Automated Generation of Intent-Based 3D Illustrations"**. In *Computer Graphics (SIGGRAPH '91 Conference Proceedings)*, pages 123–132, 1991. (cited on page 9)

[SGI93] **"Onyx Graphics Supercomputers"**. Silicon Graphics Product Information, 1993. (cited on page 11)

[SGI94] **"Indy Technical Report"**. Silicon Graphics Product Information, 1994. (cited on pages 115, 158)

[Shareef95] N. Shareef and R. Yagel. **"Rapid Previewing via Volume-Based Solid Modelling"**. *Solid Modeling*, 1995. (cited on page 70)

[Sillion89] F. Sillion and C. Puech. **"A General Two-Pass Method Integrating Specular and Diffuse Reflection"**. In *Computer Graphics (SIGGRAPH '89 Conference Proceedings)*, pages 335–344, 1989. (cited on page 131)

[Sillion91] F. X. Sillion, J. R. Arvo, S. H. Westin, and D. P. Greenberg. **"A Global Illumination Solution for General Reflectance Distributions"**. In *Computer Graphics (SIGGRAPH '91 Conference Proceedings)*, pages p187–196, 1991. (cited on pages 98, 131)

[Sillion95] F. Sillion and G. Drettakis. **"Feature-Based Control of Visibility Error: A Multi-Resolution Clustering Algorithm for Global Illumination"**. In *Computer Graphics (SIGGRAPH '95 Conference Proceedings)*, pages 145–152, 1995. (cited on pages 132, 171)

[Smits92] B. E. Smits, J. R. Arvo, and D. H. Salesin. **"An Importance Driven Radiosity Algorithm"**. In *Computer Graphics (SIGGRAPH '92 Conference Proceedings)*, pages 273–282, 1992. (cited on pages 116, 123, 131)

[Smits94] B. Smits, J. Arvo, and D. P. Greenberg. **"A Clustering Algorithm for Radiosity in Complex Envrionments"**. In *Computer Graphics (SIGGRAPH '94 Conference Proceedings)*, pages 435–442, 1994. (cited on page 132)

[Snyder92] J. M. Snyder. **"Generative Modelling for Computer Graphics and CAD"**. Academic Press, 1992. (cited on pages 5, 62, 66)

[Snyder95] J. M. Snyder. **"An Interactive Tool for Placing Curved Surfaces without Interpenetration"**. In *Computer Graphics (SIGGRAPH '95 Conference Proceedings)*, pages 209–219, 1995. (cited on page 5)

[Stonebraker93] M. Stonebraker, J. Chen, N. Nathan, C. Paxson, and J. Wu. **"Tioga: Providing Data Management Support for Scientific Visualisation Applications"**. In *Proceedings of the 19th Very Large Databases Conference*, pages 25–38, 1993. (cited on page 147)

[Strothotte94] T. Strothotte, B. Preim, A. Raab, J. Schumann, and D. R. Forsey. **"How To Render Frames and Influence People"**. *Eurographics 1994 Conference Proceedings (Computer Graphics Forum)*, volume 13 number 3, pages 455–466, 1994. (cited on pages 9, 139)

[**Stroustrup91**] B. Stroustrup. **"The C++ Programming Language"**. Addison Wesley, 1991. (cited on page 115)

[**Subramanian90**] K. R. Subramanian and D. S. Fussell. **"A Cost Model for Ray Tracing Hierarchies"**. Technical Report TR-90-04, University of Texas at Austin, 1990. (cited on page 108)

[**Subramanian91**] K. R. Subramanian and D. S. Fussell. **"Automatic Termination Criteria for Ray Tracing Hierarchies"**. In *Proceedings of Graphics Interface '91*, pages 93–91. Canadian Information Processing Society, 1991. (cited on pages 26, 108)

[**Sweldens94**] W. Sweldens. **"Wavelets, Signal Compression and Image Processing"**. In *Wavelets and their Applications in Computer Graphics*, (Ed. A. Fournier) *SIGGRAPH '94 Course Notes*, number 11. 1994. (cited on page 98)

[**Teller91**] S. J. Teller. **"Visibility Preprocessing for Interactive Walkthroughs"**. In *Computer Graphics (SIGGRAPH '91 Conference Proceedings)*, pages 61–69, 1991. (cited on page 28)

[**Teller93**] S. Teller and P. Hanrahan. **"Global Visibility Algorithms for Illumination Computations"**. In *Computer Graphics (SIGGRAPH '93 Conference Proceedings)*, pages 239–246, 1993. (cited on page 148)

[**Teller94**] S. Teller, C. Fowler, T. Funkhouser, and P. Hanrahan. **"Partitioning and Ordering Large Radiosity Computations"**. In *Computer Graphics (SIGGRAPH '94 Conference Proceedings)*, pages 443–450, 1994. (cited on page 131)

[**Thompson91**] K. Thompson and D. Fussell. **"Time Costs of Ray Tracing with Amalgams"**. Technical Report TR-91-01, University of Texas at Austin, 1991. (cited on pages 37, 91, 128)

[**Tommasi90**] G. F. M. Tommasi. **"Procedural Methods in Computer Graphics"**. PhD thesis, University of Cambridge Computer Laboratory, 1990. (cited on page 71)

[**Treisman86**] A. Treisman. **"Properties, Parts and Objects"**, volume II of *Handbook of Perception and Human Performance*, chapter 35, (Eds. Bofl, Kaufman, and Thomas). Wiley, 1986. (cited on page 50)

[**Turk92**] G. Turk. **"Re-Tiling Polygonal Surfaces"**. In *Computer Graphics (SIGGRAPH '92 Conference Proceedings)*, pages 55–64, 1992. (cited on pages 5, 56)

[**TVision95**] ART+COM. **"The T_Vision Project"**. WWW site "http://www.artcom.de/projects/terra", 1995. (cited on page 12)

[Upson88] C. Upson and M. Keeler. **"V-BUFFER: Visible Volume Rendering"**. In *Computer Graphics (SIGGRAPH '88 Conference Proceedings)*, pages 59–64, 1988. (cited on page 70)

[Upstill90] S. Upstill. **"RenderMan Companion"**. Addison Wesley, 1990. (cited on pages 98, 116)

[Vemuri94] B. C. Vemuri and A. Radisavljevic. **"Multiresolution Stochastic Hybrid Shape Models with Fractal Priors"**. *ACM Transactions on Graphics*, volume 13 number 2, pages 177–207, April 1994. (cited on page 65)

[Wallace87] J. R. Wallace and D. P. Greenberg. **"A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods"**. In *Computer Graphics (SIGGRAPH '87 Conference Proceedings)*, pages 311–320, 1987. (cited on page 131)

[Wallace89] J. R. Wallace, K. A. Elmquist, and E. A. Haines. **"A Ray Tracing Algorithm for Progressive Radiosity"**. In *Computer Graphics (SIGGRAPH '89 Conference Proceedings)*, pages 315–324, 1989. (cited on page 32)

[Ward88] G. J. Ward. **"A Ray Tracing Solution for Diffuse Interreflection"**. In *Computer Graphics (SIGGRAPH '88 Conference Proceedings)*, pages 85–92, 1988. (cited on page 124)

[Warn83] D. R. Warn. **"Lighting Controls for Synthetic Images"**. In *Computer Graphics (SIGGRAPH '83 Conference Proceedings)*, pages 13–21, 1983. (cited on page 129)

[Webspace] **"Silicon Graphics WebSpace Navigator"**. WWW site "http://webspace.sgi.com/WebSpace/". (cited on page 164)

[Wernecke94] J. Wernecke. **"The Inventor Mentor"**. Addison Wesley, 1994. (cited on pages 4, 91, 115, 116, 150, 158)

[Westin92] S. H. Westin, J. R. Arvo, and K. E. Torrance. **"Predicting Reflectance Functions from Complex Surfaces"**. In *Computer Graphics (SIGGRAPH '92 Conference Proceedings)*, pages 255–264, 1992. (cited on page 98)

[Wilhelms94] J. Wilhelms and A. Van Gelder. **"Multi-Dimensional Trees for Controlled Volume Rendering and Compression"**. Technical Report UCSC-CRL-94-02, University of California, Santa Cruz, 1994. (cited on page 68)

[Williams83] L. Williams. **"Pyramidal Parametrics"**. In *Computer Graphics (SIGGRAPH '83 Conference Proceedings)*, pages 1–11, 1983. (cited on pages 32, 54, 98)

[Wilson94] O. Wilson, A. Van Gelder, and J. Wilhelms. **"Direct Volume Rendering via 3D Textures"**. Technical Report UCSC-CRL-94-19, University of Calfornia, Santa Cruz, 1994. (cited on page 70)

[Winkenbach94] G. Winkenbach and D. Salesin. **"Computer Generated Pen-and-Ink Illustration"**. In *Computer Graphics (SIGGRAPH '94 Conference Proceedings)*, pages 91–100, 1994. (cited on page 9)

[Wloka93] M. Wloka. **"Incorporating Update Rates into Today's Graphics Systems"**. Technical Report CS-93-56, Brown University, 1993. (cited on pages 105, 134)

[Wrigley93] A. Wrigley. **"Real-Time Ray Tracing on a Novel HDTV Framestore"**. PhD thesis, University of Cambridge Computer Laboratory, 1993. (cited on page 124)

[Wu93] Z. Wu, K. Moody, and J. Bacon. **"A Persistent Programming Language for Multimedia Databases in the OPERA project"**. Technical Report, University of Cambridge Computer Laboratory, 1993. (cited on page 146)

[Yan85] J. K. Yan. **"Advances in Computer-Generated Imagery for Flight Simulation"**. *IEEE Computer Graphics & Applications*, volume 5 number 8, pages 37–51, August 1985. (cited on page 52)