Intelligent Modeling and Manipulation of Three Dimensional Objects in Computer Vision and Animation

by

Li Rong

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Systems Design Engineering

Waterloo, Ontario, Canada, 1998

0-612-60811-5

Abstract

In this thesis, the author investigates the use of Attributed Hypergraph Representation (AHR) based on category theory as the supporting framework of a generic 3D object modeling and manipulation methodology. The efficacy of the methodology is illustrated by its applications in computer vision and animation. Compared with the relational graph used in computer vision [79] and the scene graph used in computer graphics [17], AHR has the advantage that it is capable of representing multiple relations by its hyperedge structure. One of the most important contributions of the thesis is the mathematical framework established to examine AHR from a theoretical perspective.

From multiple 2D views of a 3D object or scene, range information is first computed and then a triangular mesh model is built. A net-like data structure of AHR can be configured on this mesh model. The data structure is designed to handle the transformations on the representation corresponding to the object's movements and deformations. In an attributed hypergraph, the attributes associated with the hyperedges and the vertices give it power to model arbitrary shapes with geometrical, physical or behavioral features. As a hierarchical and generic representation, AHR enables pattern matching, recognition, synthesis and manipulation to be carried out at different resolution levels or on different subsets depending on the context. Symbolic computation on knowledge represented in the format of attributed hypergraphs becomes straightforward.

From the mathematical viewpoint, a representation on a feature level with the transformations defined on it, such as physically based modeling [85] with the state transition functions, forms a category; AHR and AHR transformations form another category. Given the features of a 3D object or scene, the procedure of constructing

the AHR corresponds to the concept of *functor* in category theory, which maps one category to another one. The transformations of AHR are in the form of a set of operators defined on attributed hypergraphs, which stand for the motions and deformations of the object.

This representation is applied to various modeling and manipulation tasks on 3D objects:

- the process of motion analysis of a 3D object is the task of extracting a sequence of AH operators from the AHR of the object;
- a 3D scene can be modeled by AHR and then altered/augmented with other 3D models, by which an augmented reality can be built;
- given the AHR's of two different 3D shapes, 3D morphing can be accomplished by matching the two AHR's and then mapping the difference to a sequence of AH operators;
- model based animation of an object can be accomplished by applying a set of AH operators to its AHR.

Acknowledgements

I would like to thank my supervisor, Prof. A. K. C. Wong, for his wisdom. his guidance, his encouragement, and his patience throughout my studies.

I am indebted to the members of my committee, Prof. D. Lowe, Prof. M. McCool. Prof. J. Mcphee, Prof. O. Basir and Prof. R. Kazman, for their valuable comments and suggestions. I would also like to thank various faculty members of the Department of Systems Design Engineering, especially Dr. P. Calamai and Dr. M. Kamel, for their advice, guidance and other supports in producing this thesis.

I am grateful to Dr. Y. Wang, Dr. T. Chau, Mr. X. He, Mr. M. Konstantinidis. Mr. P. Deelstra, Mrs. A. Dietrich, Mrs. P. Polan, and everyone who has contributed to my education and research during my college years, and to everyone who has encouraged and helped me in my life.

Last, but not the least, I would like to thank my wife Judy. Without her love and support, it would have been impossible for me to accomplish such a major task.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computational modeling has been an active and productive research field for some time. It provides the theoretical fundamentals for many research topics, such as machine vision, computer animation and engineering visualization. Ever since the 1950's, it has been a goal of scientific and engineering research to produce intelligent machines that can effectively model and understand what humans see in the environment. It can make computer not only a machine for computation, but also a tool to help people exploring their world. Since the 1980's, the applications of computers are no longer restricted in traditional science and engineering areas. They have immersed into many aspects of our everyday life. With the emergence of "computer animation" and "virtual reality" technologies, which have been highly successful commercially, to develop an intelligent modeling technology for our surrounding world becomes increasingly important.

With a set of camera views and constraints about a scene, the objective of three dimensional (3D) modeling is to derive certain descriptive (structural, topological or procedural) information through image understanding. This includes the analysis and synthesis of 3D shapes, surface textures, colors, and more importantly, the 3D motions of the objects in the images. With the aid of computer vision and computer graphics technologies, the latest development in 3D object modeling has had great impact in the industrial fields as well as in our daily lives.

1.1 Applications of 3D Object Modeling

The applications of 3D object modeling have a broad range. In this section, three typical applications, namely (1) model based 3D computer vision, (2) computer animation and (3) virtual reality and augmented reality, are briefly introduced.

1.1.1 Model Based 3D Computer Vision

3D computer vision is an area that directly makes use of 3D object modeling. Its most successful applications include 3D scene understanding for autonomous vehicle steering and real-time 3D object recognition on industrial production line.

In many model based 3D vision systems, it is required to search for the best match between known 3D models and 2D perspective views of the objects. Complete or partial 3D models can be matched against 2D views for object identification, 3D pose calculation or 3D motion analysis. The system's performance normally depends on how the object is represented. In many applications, the vision system has to either have the prior knowledge of some prominent "landmarks" in the scene, or have the ability to infer relevant 3D information based on the scene from incomplete world model. This means that the modeling approach is required to record the appearance features of the scene, and most preferably, to support knowledge processing such as learning. In this thesis research, 3D objects are represented in a symbolc way, by which, pattern learning and recognization can be performed within a unified framework.

1.1.2 Computer Animation

Over the years, computer animation has been making steady and rapid progress, largely benefited from its great success in the entertainment fields. In a typical animation system, under the control of the operator (called the *animator*), rigid or deformable multi-body is animated from its geometrical models according to the specifications of kinematic and dynamic constraints. An intelligent modeling approach minimizes the interactions from the operator. It should allow automatic derivation of the local motion/deformation form and the motion trajectory.

Therefore, a good modeling methodology for computer animation will have at least the following components:

- appearance modeling which includes geometric shapes, textures and colors;
- kinematics modeling which includes manipulations such as rotation and translation of the objects as well as collision detection;
- dynamics modeling which specifies the objects' physical parameters, such as mass, weight, inertial, compliance, damping factor, etc.

These features are only the minimal requirements. There may be extra features for more complex subjects such as the so called "intelligent agents" and "emotional agents". To incorporate and operate on those features, there is a need for a generic representation for modeling, which empowers the operator to animate the graphical objects at different information levels with symbolic operations.

1.1.3 Virtual Reality and Augmented Virtual Reality

Virtual reality (VR) is a research area emerging from the late 1980's. Although the basic concepts have been proposed earlier, it did not catch enough interest until the maturity of computer graphics and visualization technologies. The basis of VR is not only to present to but also to "immerse" users with an imaginary and computer generated "virtual world". Developed as an extension of VR, augmented virtual reality (AR) lets the user interact with a "real" world, on which there is also computer graphics superimposed. In AR, it would appear to the user that both real and virtual objects coexist [5].

In the theoretical research of VR and AR, much of the focus is on designing an appropriate 3D modeling methodology. In VR, an ideal modeling approach should enable real-time performance and vivid presentation. It is required that the object's geometrical features, together with the textures, colors and physical parameters be registered in a unified representation. In AR, the methodologies to construct the virtual part can be directly adopted from those of VR's. However, the processing and the integration of the real part with the virtual parts are not easy tasks. Seamless integration depends on how the real scene and the virtual world are modeled.

1.2 Problem Definition

1.2.1 Motivation

In-depth studies of the human vision system have shown that the analysis of dynamic scene involves both low-level processing at the retina and high-level processing in the brain [20] . Burr's experiments showed that the human vision system capture both high-level structures and low-level entities [21]. A successful machine vision system should probably also have these characteristics. Unfortunately, most current systems do not fully satisfy this requirement. They have object models that are difficult to operate with at symbolic or knowledge level. The inflexibility of the pre-chosen models results in difficulties in manipulation and poor approximations. Furthermore, there is no way to "learn" unknown patterns from the sensed data.

Previous modeling techniques mainly looked after geometrical appearance or physical features. With incomplete modeling, most tasks were accomplished with enormous human assistance. To incorporate various features into operations and to automate the process, it is required to build a generic representation to enable symbolic operations for the animation of a virtual 3D world with different types of information.

1.2.2 Research Objectives

There is still no agreement on the criteria for an ideal 3D representation for modeling. However, the following criteria [11] would be at least a necessary subset:

- **computable**: the computational complexity should be of low order in time and space, and an incomplete representation must still be computable;
- **stable**: small local noises in the data should only introduce small variation in the representation;
- **unique**: a given object must have a unique representation;
- **complete:** within a problem domain, for each and every object there is a corresponding representation;

- **composite:** the natural recursive part-whole composition structure in the objects should be explicit in the corresponding representation;
- **invariant:** an object representation should be invariant under geometric transformations;
- **generic:** it should enable pattern matching, recognition and reconstruction to be carried out at various resolution levels depending on the context.

Among the above criteria, the last two are of particular importance. First, the representation has to accommodate the dynamic characteristics such as translations, rotations and deformations. Second, it is necessary that a representation support a generic class of objects as well as the specific ones.

In the research of visual representations, there is a growing interest in graph-like representations. In general, graphs provide a straightforward and flexible method to describe primitives and their relations. In computer vision, many successful results from graph based representations have been reported since the mid-1980's [79, 99, 102, 103]. Among the pioneers are the concept of *relational graph* in 3D object representation [79], and attributed hypergraph (AH) modeling [102] for 3D recognition and model synthesis. In graphics modeling, trees and graphs have been applied to represent hierarchical structures [17]. The author believes that for both computer vision and animation, an attributed hypergraph is ideal as a generic representation and an effective manipulation tool.

The objectives of this thesis research are:

- to study the existing methodologies in 3D object modeling;
- to establish a mathematical framework for attributed hypergraph representations (AHR), and to examine them from a mathematical perspective;

- to explore the algorithms that build, manipulate and augment AHR's;
- to develop an efficient algorithm that can automatically control and generate 3D morphing and realistic animations.

The ultimate goal of the research presented in and extended beyond this thesis is to implement a system that integrates machine vision and computer graphics technologies based on the proposed 3D object modeling. Such a system can be illustrated by Figure 1.1, in which the modules with solid borders are the focuses of this thesis.

1.3 Thesis Outline

This thesis consists of six chapters. This chapter is the general introduction to the research. Chapter 2 reviews the literature in related topics. Instead of focusing on the theoretical foundations, much emphasis has been placed on the applications in machine vision and computer graphics.

Chapter 3 introduces and elaborates the mathematical aspect of the hypergraph based object modeling methodology with theoretical support from category theory. Some terminologies in graph theory and category theory are introduced, followed by the formalization of the *attributed hypergraph representation* (AHR), the dynamic data structure for AHR and the AH operators. An AHR is constructed from feature based triangular meshes of a 3D object, where the features can be geometrical, kinematic, physical, behavioral ones or their combinations. The construction of the AHR corresponds to a functor in category theory.

Chapter 4 presents the applications of AHR in augmented reality, which are primarily computer graphics tasks that utilize 3D vision technologies. Traditional

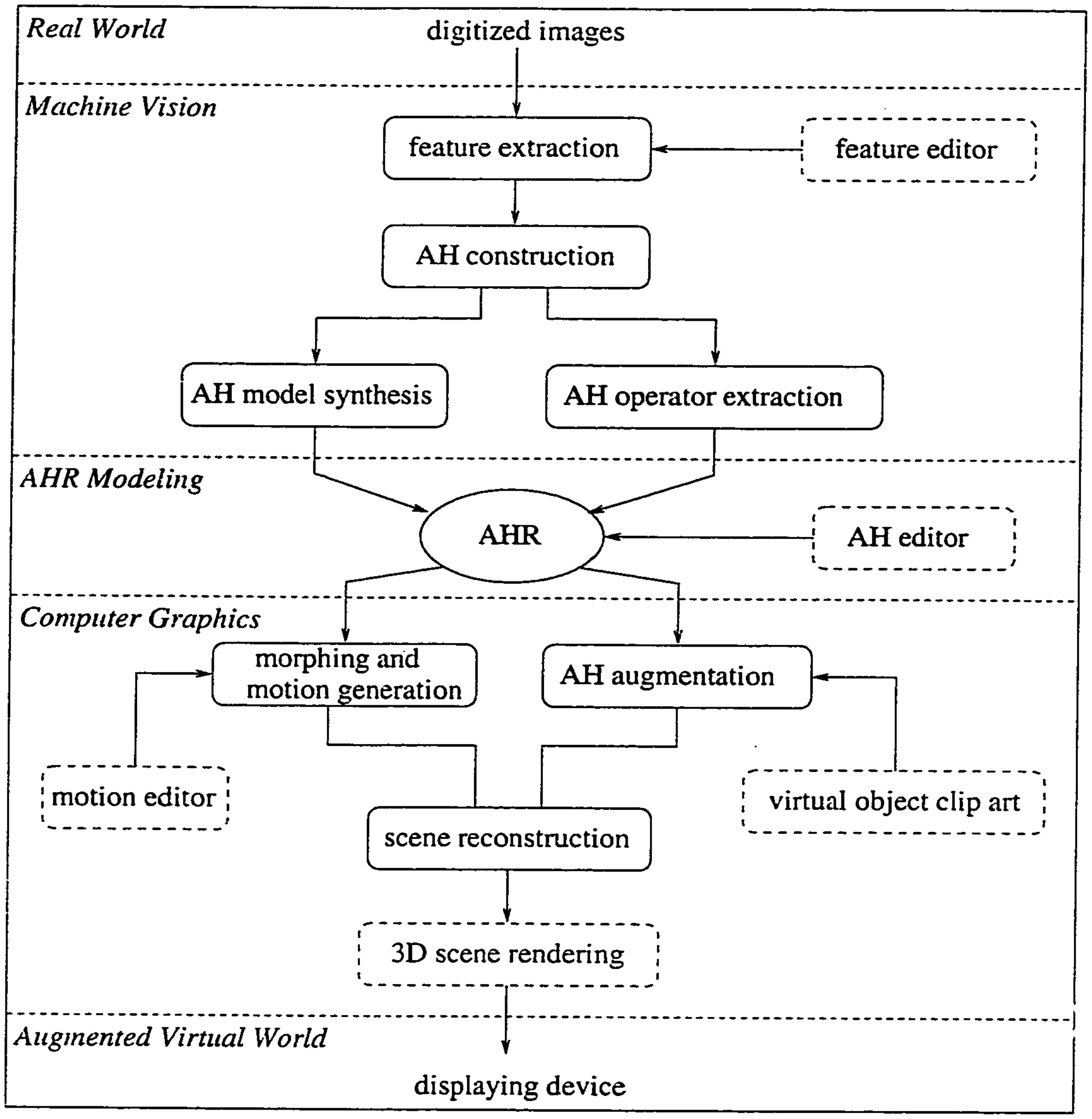


Figure 1.1: The integrated system of machine vision and computer graphics technologies.

3D computer vision techniques, such as camera calibration and stereo triangulation, provide the basis of the proposed AR system. From the available 3D visual data, an AHR can be constructed to form a framework on which high-level processes, such as automatic augmentations, are easy to perform. The experiments show that AHR is very flexible for 3D scene understanding, reconstruction and augmentation.

Chapter 5 focuses on AHR's applications in 3D morphing and intelligent animation. Before the applications are addressed, the optimal subgraph isomorphism algorithm for AHR's is elaborated. Different 3D shapes or different statuses of a dynamic scene are represented by attributed hypergraphs. They can be matched by searching for the optimal common subgraph. Then, sets of AH operators are extracted from the discrepancies. The AH operators register the qualitative and quantitative changes of the motions that conjoin the two different AH's. Automatic morphing or animation can be performed by applying the AH operators on the source AH.

Chapter 6, the conclusion of the thesis, begins with a general comment on the new methodology. It then presents the anticipated contributions of the research, some possible improvements, and finally the trends of future work.

Chapter 2

Review of Literature

2.1 3D Object Modeling for Computer Vision and Graphics

The representation of 3D shapes has preoccupied computer vision and graphics researches for several decades. Before the emergence of computer animation, the research mainly focused on the modeling of rigid shapes. Despite the large body of work. most techniques lacked the flexibility to model non-rigid motions. Only after the mid 1980's were a number of modeling methodologies proposed to solve the problems of deformations. In this chapter, we focus on the underlying mathematics and the process involved in 3D object modeling. We will first introduce the principles of object modeling and then review different types of modeling methodologies.

2.1.1 Principles of 3D Object Modeling

An effective 3D object modeling methodology should characterize the object's features under different circumstances in the application scope. In the early research, people focused on appearance modeling since the objects studied in vision and graphics applications were simple and stationary. Nowadays, with the development of dynamic vision and computer animation, simple unilateral modeling can no longer satisfy the requirement. A complete 3D object modeling should at least comprise the following components [19]:

- *Geometrical Modeling*

  This is the basic requirement for any vision or graphics system. It describes an object's geometrical properties, namely, the shape (e.g., polygon, triangle or vertex, etc.) and the appearance (e.g., texture, surface reflection or color).

- *Kinematic Modeling*

  It specifies an object's motion behaviors which are vital for dynamic vision and animation. A 4×4 or 3×4 homogeneous transformation matrix can be used to identify translations, rotations and scalings and as the base for collision detection.

- *Physical Modeling*

  Physical modeling is required for complex situations where deformations and collisions are involved. Objects can be modeled physically by specifying their mass, weight, inertia, compliance, deformation parameters, etc.. These features are merged with the geometrical modeling along with physical laws to form a realistic model.

- *Behavior Modeling*

  This is the least studied aspect of modeling. In intelligent modeling, an object can be considered as an "intelligent agent" which has a degree of intelligence. It can actively response to its environments based on certain rules.

In the following, the modeling methodologies in the literature are reviewed and classified into three categories, namely, continuous modeling, discrete modeling, and graph based modeling.

2.1.2 Continuous Modeling

This type of modeling approximates either the whole or a functional part of the 3D object by a variation of geometrical primitives, such as blocks, polyhedrons, spheres, generalized cylinders or superquadrics. These geometrical primitives can be expressed as continuous or piecewise continuous functions in 3D space. Kinematic and physical features can be easily combined with the geometrical shapes. Among the large body of the geometrical primitives, generalized cylinders and superquadrics are the popular ones since they could easily handle deformations.

Barr is considered as one of the first to "borrow" the techniques from linear mechanical analysis to approximate visual 3D objects [9, 10]. He defined the angle-preserving transformations on superquadrics. Although the original approach was only for computer graphics, it is also very useful in vision tasks and has led to fruitful results.

As a dynamic extension of superquadrics, the "deformable superquadrics" [88] proposed by Terzopoulos *et. al.* is a physical feature based approach. It fits complex 3D shapes with a class of dynamic models that can deform both globally and locally.

The model incorporates the global shape parameters of a conventional superellipsoid with the local degrees of freedom of a spline. The local/global representation power simultaneously satisfies the requirements of 3D shape reconstruction and 3D recognition. In an animation task, the behaviors of the deformable superquadrics are governed by motion equations based on physics. In 3D model construction, the model is fitted with 3D visual information by transforming the data into forces and simulating the motion equations through time.

In animation tasks, it is easy to detect, attach and apply geometrical, kinematic and physical parameters to continuously modeled objects. However, for behavioral features, it is difficult since the model lacks a symbolic structure as the base to fit in behavioral languages. Furthermore, for many real world objects, approximation by pre-defined primitives such as generalized cylinders or superquadrics is impossible.

2.1.3 Discrete Modeling

A variety of computer vision applications involve highly irregular, unstructured and dynamic scenes. They are characterized by rapid and non-uniform variations in spatially irregular feature densities and physical properties. It is difficult to model such objects from any of the aspects mentioned before with continuous elements. This difficulty stems from the unpredictable behaviors of the objects. Discrete modeling is able to approximate the surfaces or volumes of this kind of objects by vast patches of very simple primitives, such as polygons or tetrahedrons.

Since most graphics applications use polygons as the fundamental building block for object description, a polygonal mesh representation of curved surfaces is a natural choice for surface modeling. Several approaches are available to generate polygonal meshes from explicit boundary representations of 3D objects and their surfaces

[92]. Polygonal approximation of sensory data is relatively simple to carry out. Sampled surfaces can be approximated to the desired precision [56]. Physical and kinematic features can be associated with either a single element (a polygon) or a group of elements (a patch of polygons), which offers more flexibility.

Triangular meshes are a special case of polygonal meshes. They have been recognized as a powerful tool for surface modeling due to their simplicity and flexibility. Furthermore, the abundance of algorithms to manipulate triangular meshes encourages and facilitates their use in many general vision and graphics applications [34, 39, 74, 83]. They provide fast preprocessing, data abstraction and mesh refinement techniques.

Based on the triangular mesh, Terzopoulos and Waters have successfully attached physical constraints on the human facial models [89]. From sequences of facial images, they built mesh models with anatomical constraints. An impressive advance of the methodology is that it has the capability to model the behavioral features. With the support of anatomical data, different emotions can be modeled and applied to arbitrary human faces.

The main drawback of mesh based methodologies is that they lack the high level structure to control the modeling or to perform symbolic operations. The polygonal primitives are unstructured and contain only information on local features. When applied to dynamical cases, even though geometrical modeling can be highly precise, abstracting high level information from the data is still problematic.

2.1.4 Graph Based Symbolic Modeling

In graph based approaches, a complex object is usually represented explicitly by a set of primitives and the relations among them in the form of a graph. If the primitives are several regular blocks such as cylinders, cubes or superquadrics, it can be viewed as a clone of continuous modeling. Similarly, if the model consists of vast number of primitives that are discrete polygons or polygonal meshes, it is the extension of discrete modeling.

The graph representation was first put forward in 1970 [80] neither for computer vision nor for graphics applications, but for the script description of a scene in artificial intelligence. In the early 1980's, Shapiro applied relational graphs for object representation [79]. The quadtree (for surface) and octree (for solid) encoding algorithm given by Meagher [66] can be viewed as special cases of graph representation, since they form trees (directed graphs) as the hierarchical structure for representation. Later, Wong and his colleagues presented random graph and attributed hypergraph as geometry and knowledge representation for computer vision. In 3D model synthesis, random graphs were applied to describe the uncertainties brought by sensors and image processing [103]. In [102], attributed hypergraph model was constructed based on model features. The representation had a four-level hierarchy that characterizes: 1) the 3D geometrical model features; 2) the characteristic views induced by local image/model features, each of which contains a subset of the model features visible from a common viewpoint; 3) a set of topological equivalent classes of the characteristic views; and 4) a set of local image features. Domain knowledge could be imposed on the representation for various forms of decision making and reasoning.

Since graph based modeling approaches introduce the concepts of *primitives* and their *relations*, it is straightforward to build a hierarchical representation. At lower hierarchies, geometrical, kinematic and physical features can be encoded as the primitives and the attributes associated with them; while at higher hierarchies, the entities such as edges and hyperedges can be used to represent symbolic information. In a graph structure, it is handy to perform symbolic operations. With the aid of machine intelligence technologies, domain knowledge can be learned and later recalled together with other data. However, graph based approaches lack the representational power for dynamic objects. Although they have the potential to handle deformations, up to now, the graph based systems have only been applied to rigid object modeling, mainly due to the rigid structure of graph and the lack of definitions for transformation operators on graphs.

In this thesis, the attributed hypergraph representation (AHR) is applied to develop a 3D modeling methodology based on triangular mesh approximation of the object surfaces. Surface modeling approaches were chosen over solid modeling because the modeling approach should fit in both computer vision and computer graphics applications, while most vision sensors provide surface data only. A net-like data structure is designed to handle the dynamic changes in the representation corresponding to the object's movements and deformations to overcome the inflexibility of the graph structure.

2.2 Augmented Reality

Augmented Reality (AR) is an extension of *Virtual Reality* (VR) technologies. In the past five years or so, VR technologies have attracted a great deal of research interests as well as the media attentions due to their great success in commercial applications. The basic idea of VR is not only to present to the users but also to "immerse" them in an imaginary, computer generated "virtual world". Although many different systems have been developed, they all share a common drawback: the user is cut off from any view of the real world [19, 43].

Augmented reality emerged as a solution to the above problem. In an augmented reality system, there are sensory devices that capture the features of the real world. It then embeds or superimposes them into a computer graphics environment. In other words, instead of merely replacing the real world by the virtual world, it supplements the virtual world with the real world features or vice versa. Thus, it appears to the user that the features captured from the real world and those created by computer graphics coexist [5].

2.2.1 Characteristics of Augmented Reality

This section reviews the common features of the current AR systems in sensing, scene augmentation and 3D registration. The problems of AR algorithms are also addressed.

Sensing

AR systems have more strict requirements on the sensing devices than those of VR systems. Specifically, AR requires the sensors to have more flexibility and higher accuracy at wider sensing range.

In VR, the sensors are only used to track the positions of the user's eyes/hands by certain pointing devices (e.g., 3D mouse and data glove). AR systems, however, have a greater variety of input sensing devices to capture real world data and to track the viewpoints. These devices may include CCD cameras, laser range finders, CT, MRI or ultrasound sensors and more. Conceptually, for the purpose of visualization, anything that can be detected by machines might be transduced into the AR system in future applications [5].

Range image data are very important for the current AR applications. A computer system has the exact 3D information of the virtual elements but may not know where all the real elements are in its environment. Furthermore, in many applications, the environment is not static. In other words, the real elements must be tracked in real time. Normally full 3D data of the real scene are required. However, in many cases, for real-time performance, the minimal requirement of the real elements would be the range data. A direct solution to obtain 3D or range data is to use a laser range-finder. More cost effective methodologies are using computer vision techniques, such as *shape from stereo* [61], *shape from contour* [93] *shape from motion* [53] and *shape from shadow* [24], with inexpensive CCD cameras.

The accuracy required for sensing is driven by the accuracy needed for visual registration (refer to Section 2.2.1). Approaches directly adopted from VR systems cannot meet this specification since they are sensitive to the conditions of the AR environments. For example, inertial force based sensors drift with time, magnetic force base sensors are sensitive to metal disturbance, and sonar sensors suffer from noises. Laser range finders would solve the problem with the sacrifice of high cost. Another solution is the traditional optical sensors. The main problems of optical technologies are: (1) lens distortion, and (2) difficulty in calibration. Many attempts have been made to correct the optical distortions of cameras [1, 91], and to certain extent, they have succeeded in compensating the lens distortions and achieving sub-pixel accuracy. Optical sensors seem to be most promising due to the trends towards high-resolution digital cameras and real-time photogeometric technologies.

Augmentation

Current work on augmentation in AR has focused on adding virtual objects to a real environment, although the capability should also include removing real objects from a real scene.

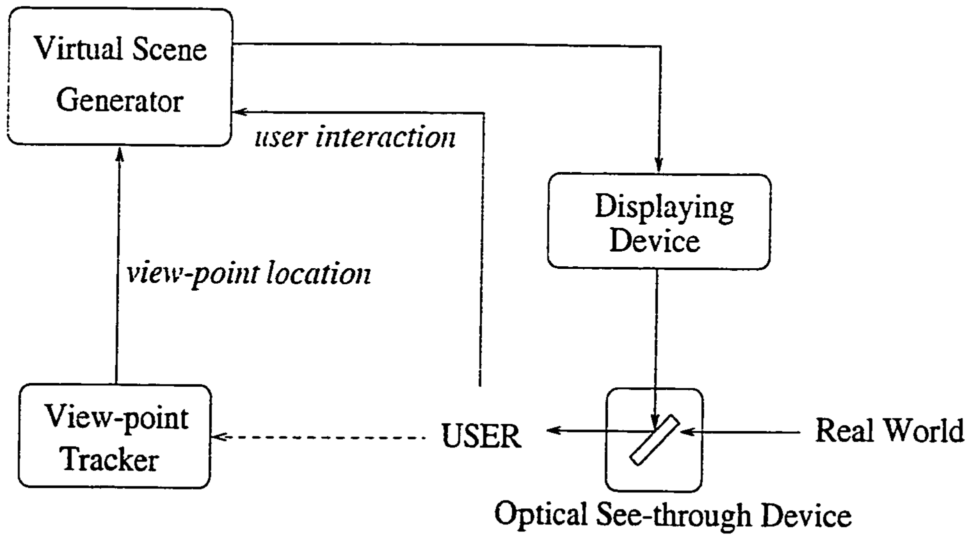


Figure 2.1: Optical see-through device used for augmented reality.

A feasible implementation of adding virtual objects is the optical combination via an optical device with semi-transparent lenses, called the "see-through" device, such as the Head-Mounted Display (HMD) or *Head Up Display* (HUD) [95]. This device places optical combiners in front of the user's eyes. The combiners let light in partially from the real world, while at the same time reflect also light from a monitor that displays graphical images. The result is a combination of the real world and the virtual world displayed by the computer. Figure 2.1 illustrates the configuration of the see-through system.

Though the see-through system is fast and cost effective, it has a serious drawback in that the part of the real world is not changeable. Therefore, it can only be presented exactly as what it is in real world. Consequently, there is no way to handle subtle cases such as removing real objects, presenting virtual occlusions or virtual lighting effects. Furthermore, as discussed later, it is almost impossible to calibrate the registration errors in this type of AR system.

An ideal augmentation (also referred to as "integration") can be accomplished

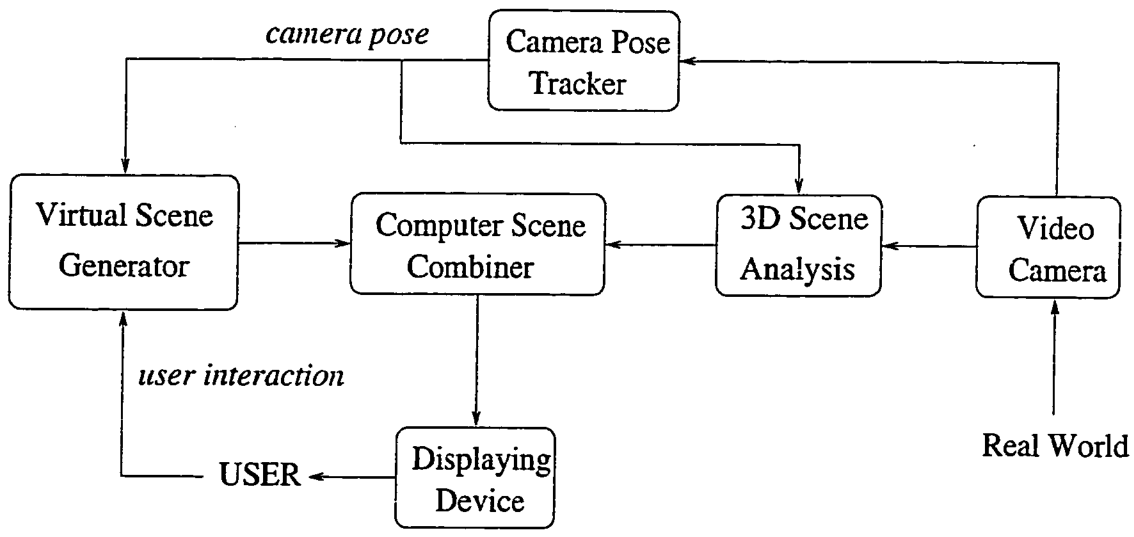


Figure 2.2: Improved see-through device with computer aided video combination for augmented reality.

by using the intelligent computer algorithms (refer to Figure 2.2) called "reconstruction based system". Images from real scenes should be first acquired, digitized and input into computers. They are analyzed to extract appropriate 3D information. Then, the scenes reconstructed from the real world are blended with the virtual elements. Finally, the combined scenes are adjusted with possible texture and lighting changes to the real world features or virtual domains and projected to the viewing plane. Though the reconstruction based methodology is much more elaborate and time-consuming compared to the optical one, its advantage is that with 3D visual information, we can have: (1) correct computation on occlusions; (2) robust and accurate determination and tracking of the camera motion; and (3) flexible and precise 3D placement of real and virtual objects/elements.

Registration

One of the most fundamental problems currently limiting AR applications is the registration problem. Though similar problem appears in VR applications, it is relatively easy to solve compared with that of AR [4]. In AR applications, the objects in the real and virtual worlds have to be properly aligned with respect to each other. Therefore, accurate registration is of great importance.

There are five main sources of registration errors:

- optical distortions;
- errors from the tracking system:
- mechanical misalignments;
- inaccurate viewing parameters;
- system delays in real-time applications.

Most current AR implementations register images solely based on non-visual tracking systems. Typically, there are sensors that track the locations of a number of specially designed landmarks. However, as pointed out by Bajura and Neumann [7], this kind of registration is similar to an "open-loop" controller. The system has no feedback on how closely the real and the virtual elements match. Therefore, it is impossible to compensate the errors.

In an optical see-through system, the loop cannot be closed since there is no way to obtain feedback from the optical combiner. However, in a reconstruction based system (refer to Figure 2.2), computer vision technologies can be applied to calibrate the registration errors. Since in such systems digitized images of the real environment are available, it may be possible to detect features of the scene and use them to enforce the registration. This can be called a "closed-loop" approach since the features from the images provide a mechanism to include feedback into the system.

Calibration of the registration error is not an easy task. The feature detection and matching must run in real time and must be robust. Thus, the bottleneck falls into a computer vision problem. Improving computer vision algorithms will greatly help to satisfy the above requirements of AR systems even without special hardware and sensors.

2.2.2 Applications of Augmented Reality

Though some critical problems in the AR technologies are far from being solved, there are already fruitful applications in many areas.

Medical Science

During the past ten years, research in medical science has acted as one of the main sources that pushed the development of AR and VR technologies. Doctors can use AR as a visualization and training aid for surgery. Current medical imaging tools, such as MRI, CT or ultrasound provide excellent data for building 3D data-set about the inside of a patient in real time, whereas in the augmented scene, virtual elements can be attached to label the tumor or to add references. Surgeons can use augmented scenes on 3D displaying devices to guide any operations that may require high precision.

There are several projects exploring this application area. In [6], AR has been applied to scan the womb of a pregnant woman with ultrasound sensors and generating 3D views of the fetus. In MIT's AI Lab. [44] and other research institutes, there are similar projects on AR systems based on MRI and CT data. The world's first commercial image-guided surgery system is the "Viewing Wand" (ISG Technologies, Toronto) [52] which has been used for Elizabeth Taylor's brain surgery. The system can take both CT and MRI sources to collect 3D data and then apply measuring, annotation and surgical path planning to guide the surgery.

Industries

As in medical research, AR is quite productive in manufacturing fields. AR technologies in Computer Aided Manufacturing (CAM) can help people to assemble, maintain and repair complex machinery. In an industrial training system. AR can help a novice to operate machines if the step-by-step instructions are provided as 3D drawings superimposed upon the actual scene instead of the normal texts and pictures. There are several research prototypes in this area. Currently. a group at Boeing is developing an AR system to guide a technician in building a wiring harness electronically [81].

Tele-operation of robots is another area that AR technologies find enormous applications. It is normally a difficult problem when a robot is far away with long delays in the communication link. Therefore, instead of controlling the robot directly, it may be preferred to control a virtual version of the robot. The user plans and specifies the actions of the remote robot by manipulating the local virtual robot. With AR technologies, the virtual robot can be put into the pre-modeled real scene. Task/path planning can be performed first on the virtual version to allow the operator to examine the results before the actual commands are sent to the real one. The researchers at the University of Toronto have built a system called "Augmented Reality through Graphic Overlays on Stereovideo" (ARGOS).

It has demonstrated that stereoscopic AR is an easier and more accurate way of path planning than traditional monoscopic interfaces [32].

Entertainment

In the recent years, entertainment industry has brought huge profits with the applications of the latest computer animation technologies. From *Terminator II* to *Space Jam*, virtual actors had been put in a real scene, interacting with real actors, however, by off-line computer animations. In SIGGRAPH '95 exhibitions, researchers have shown systems that were able to merge real actors with virtual backgrounds in real time. Though the virtual element involved were still not complex, it is a step towards developing real-time AR gaming systems. It will not be long before the current virtual reality game systems (e.g., the 3D shooting game) could be replaced by the more exciting AR systems.

2.3 3D Morphing

2.3.1 3D Morphing versus 2D Morphing

Techniques that transform one 2D shape into another (called 2D object morphing) have gained widespread use since the late 1980's, mainly due to the high demand of the animation technologies in the entertainment industry. In recent years, with the advance of virtual reality applications, the extension of morphing technologies into 3D world, including the ability to animate 3D objects during their shape transformations, became increasingly important.

3D morphing overcomes the following shortcomings of 2D morphing [58]:

- In 3D morphing, creating the morph is independent of the viewing and lighting parameters. Hence, we can create a morph sequence once, and then project it with various camera angles and lighting conditions during rendering. In 2D morphing, a new morph must be recomputed every time we wish to alter our viewpoint or the illumination of the 3D scene.
- 2D techniques, lacking the information on the model's spatial configuration, are unable to correctly handle certain change in the environment. It will fail when occlusions among multiple objects in the scene should change during the morphing. Two examples of this type of artifacts are: (1) shadows and highlights fail to match the shape changes occurring in the morph; (2) when a part of the 3D object is not visible in the original 2D image, it cannot be made to appear during the morphing.

2.3.2 Problems of 3D Morphing

In general, 3D morphing requires true 3D continuous deformation rather than just warping of individual slices. In order to produce a realistic and smooth morphing effect, it is the model or the representation, instead of the individual 2D projective view of the 3D object, that should be transformed. By using computer animation techniques such as keyframing, transforming 3D models as opposed to images allows objects to be animated independently of the shape deformation.

Therefore, three fundamental problems that are closely related to each other have to be solved for 3D morphing:

1. *Modeling:*

   For general 3D objects, it is necessary to construct 3D deformable models that are appropriate for generating smooth shape metamorphosis.

2. *Transformation:*

   To transform one shape to another. a mapping between the two shapes has to be first established. Then from the mapping, we need to find the interpolation between the two shapes to form the continuous transformation.

3. *Control:*

   In most applications, free-form morphing is not preferred. There have to be certain kinds of constraints that control the morphing path. Therefore, it is important to allow different types of constraints and to associate them with the 3D deformable model.

2.3.3 3D Morphing Methodologies

Many methodologies have been proposed to solve the 3D morphing problem [16].

One of the first proposed 3D morphing methodologies is based on spatial modeling. It models the 3D objects by labeling the 3D geometrical key shapes and associating them according to their inter-relations. Many early attempts with spatial modeling are the kind of "brute force" approaches [84]. They essentially require the user to specify, for all points or for a number of key points in the starting model, the corresponding points in the ending model. Hong *et. al.* [47] proposed an an automatic correspondence approach for polyhedron based modeling by matching the faces of the objects whose centroids are the closest. Bethel and Uselten [13] proposed an algorithm that adds degenerate vertices and faces to two polyhedrons until a common topology is achieved. Kent *et. al.* [55] also presented a similar methodology by merging the topological structures of two 3D polyhedral models into a common vertex/edge/face network.

In general, spatial modeling based techniques are useful for simple polyhedral and cylindrical objects. When performing morphing, the user operates on the representation space by interpolation regardless of the individual properties of different parts of the objects. Other than calculating the geometrical distance, it is hard to assign any reasonable constraint to the morphing. Thus, the interpolation could be arbitrary and may not look realistic.

There are also a number of research attempts on morphing using frequency representations (e.g., avelet) of 3D objects [46, 51]. This class of methods first transform the 3D object shapes or volumes into a frequency domain, where the morphing is performed by interpolating the schedules across the sub-bands. The intermediate interpolations are transformed back to the spatial domain to form the shape deformation.

Compared with the spatial modeling based methods, these algorithms operate on the feature space where the features are the spectra of Fourier transform or wavelet transform. There is no need to match the keypoints of the two objects based on their shapes. However, a matching between the two spectra is required. Finding the matching of two spectra is not much easier than finding the matching of two shapes. Researchers have discovered that linear interpolation between two transformed data-sets in frequency domain was not satisfactory since it may yield discontinuities during the morphing. A solution proposed in [51] is to use a 3-step schedule: (1) the high frequencies of the initial model are gradually removed; (2) the low frequencies are interpolated; (3) the high frequencies of the final model are gradually added. This approach actually blurs the possible discontinuity by rounding the high frequency part where human eyes are not sensitive. It does not fundamentally solve the discontinuity problem.

Another class of algorithms which gradually gained sufficient interest in the recent years are given in the context of physical feature based modeling [10]. They use physical simulation to obtain realistic shapes and motions of a deformable object. The pioneers of this type of technology are Terzopoulos and his colleagues [85, 86]. They assigned physical properties to the objects to be deformed, such as mass and stiffness. The morphing can be considered as the result of certain external and internal forces. Then a simulation of the physical process will calculate the deformation. Later, Witkin and Welch presented a simplification of this model [97].

The technique is very promising. Since the morphing is based on a physical simulation, there is no need for the operator to specify correspondences and the morphing result looks natural. However, it too has some drawbacks. First, it is difficult to determine the appropriate external and internal forces for arbitrary specified morphing. Given two statuses of a deformable object, there may be infinite sets of forces that could deform the initial shape to the final one. Another problem is the high computation cost. Currently, the partial differential equations in the simulations are mostly solved by using a finite element method. It does not enable real-time interactive design. Furthermore, the representation of the objects is normally non-general (such as superquadrics) to simplify the complicated physical computations. Since the method relies on physical simulations solely, few geometrical and topological features of the objects are considered for the morphing. Therefore, it is hard to pose any other constraints to secure and maintain intelligent control.

In this thesis, attributed hypergraph representation (AHR) is used to model the 3D object. Morphing is generated on the representation level and then transformed back to the spatial field. Since the modeling scheme is feature based where the features are represented as the attributes in the hypergraphs, physical simulation can be accomplished if the features are assigned as physical parameters.

2.4 VRML

2.4.1 What is VRML?

VRML is an acronym for the *Virtual Reality Modeling Language.* Though it is a commercially driven product rather than a scientific research prototype, it is included in the survey due to its impressive design of the architecture (refer to Section 2.4.3), which has helped to build the data structure of AHR modeling in this thesis.

Strictly speaking, VRML is not, as it was named, a modeling language for virtual reality. It is a subset of graphical modeling language focusing on the applications in 3D Web publication. As mentioned earlier, an object modeling approach would contain much richer modeling primitives and mechanisms, with not only geometric properties, but also physical, kinematic and behavioral features. VRML provides only a bare minimum of geometric modeling, however, with numerous features far beyond the scope of modeling. Due to the fact that it adopts a graph-like data structure to represent 3D objects and that it has been widely accepted as the common 3D data format on the Web, it is important in the literature review on 3D modeling.

In [23], the answer to "*what is VRML*" is as follows:

- VRML is a 3D interchange format. It defines most of the commonly used semantics in 3D applications such as hierarchical transformations, light sources, viewpoints, geometry, animation, fog, material properties, and texture mapping.
- VRML is a 3D analogy to HTML (*HyperText Markup Language*). This means that VRML serves as a simple, multi-platform language for publishing 3D Web

pages. The use of 3D Web publishing is to solve the problem of intensive interaction, animation, user participation and exploration beyond what is capable with normal text and 2D images found in HTML.

- VRML provides a technology to integrate 3D shapes, 2D images, text and multimedia into a coherent model, which presents a natural user interface that supports traditional 2D desktop models as well as the 3D extension.

2.4.2 History of VRML

In 1989, a project named *Scenario* was started at Silicon Graphics, Inc. to design and build an infrastructure for interactive 3D graphics applications. The two original goals were: (1) to build a development environment that enabled the creation of a wide variety of interactive, distributed 3D applications: and (2) to use this environment to build a new 3D desktop interface. The early phase of the project concentrated on designing and building the semantics and mechanism for the foundational framework. In 1992, the Iris Inventor 3D toolkit was released as the first product of these efforts. Iris Inventor was a C++ toolkit that defined many of the semantics found in VRML today. An important part of the Inventor toolkit was the object oriented data structure and the external file format used for 3D objects. From the very beginning, the Inventor file format was designed to be lightweight and easy to use. In 1994, a major revision of the Inventor was released. It was called Open Inventor [69] because it was portable to a variety of platforms and it was based on Silicon Graphics' OpenGL library. The reference manual [96] describing the internal data structure of 3D objects and the file format in the Open Inventor toolkit were eventually used as a guidance in the writing the first draft of the VRML 1.0 specification.

In late 1994, a VRML mailing list was created on the Internet and then a call for proposals of a formal specification for 3D on the World Wide Web (WWW) was issued. Due to the obvious suitability of Open Inventor, it was voted as the working document of VRML, by choosing the fundamental elements of the Inventor file format and adding a couple of necessary WWW features (such as *anchor* and *inline* nodes). In October 1994, at the Second International Conference on the World Wide Web at Chicago, the first VRML specification was published [106].

During the first half of 1995, the VRML 1.0 specification underwent a variety of fixes and clarifications, but was functionally unchanged. However, an obvious shortcoming of VRML V1.0 is that it was missing some key features such as animation, interaction, and behavior. A significant revision was needed. In January 1996, a request for proposals of VRML 2.0 was issued to the VRML society led by Silicon Graphics, Inc. and Sony Corporation. In August 1996 at SIGGRAPH '96 at New Orleans, the first version of the VRML 2.0 specification was released [17].

At the July 1996 meeting in Kyoto, the International Standards Organization's (ISO) JTC1/SC24 committee agreed to publish the August 1996 version of VRML 2.0 as Committee Draft (CD) 14772. The Draft International Standard (DIS) text was submitted in April 1997. This draft of the VRML specification is known as "VRML97". The VRML97 specification was published electronically as an HTML document and marked the first time that an ISO standard has been so published [94].

2.4.3 An Overview of VRML

The following is a brief overview that describes the major features of the current VRML 2.0.

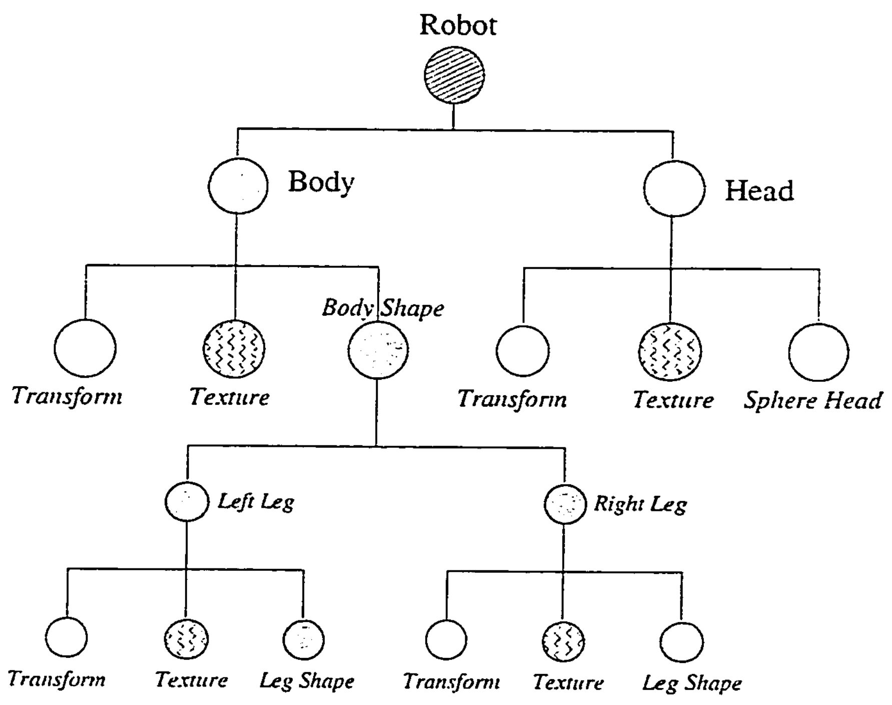


Figure 2.3: A partial scene graph of a virtual robot with a head and two legs.

Scene Graph Structure

This is the most important feature of VRML as a 3D modeling language. VRML files describe 3D objects and worlds using a hierarchical scene graph. An example of the partial scene graph of a virtual robot is illustrated in Figure 2.3. Entities in the scene graph are called nodes. VRML 2.0 defines 54 different node types, including geometry primitives, appearance properties, sound and sound properties, and various types of grouping nodes. Nodes store their data in fields, and VRML 2.0 defines 20 different types of fields that can be used to store everything from a single number (the *SFFloat* field type) to an array of 3D rotations (the *MFRotation* field type).

The VRML scene graph is a directed acyclic graph. Nodes can contain other nodes (some types of nodes may have "children") and may be contained in more than one node (they may have more than one "parent"), but a node must not contain itself. The scene graph structure makes it easy to create large worlds or complicated objects from sub-parts.

Event Architecture

VRML 2.0 defines an event or message-passing mechanism by which nodes in the scene graph can communicate with each other. Each node type defines the names and the types of events. Instances of an event may generate, receive, or *route* statements and define event paths between event generators and receivers.

Sensors

Sensors are the basic user interaction and animation primitives of VRML. The *TimeSensor* node generates events as time passes and is the basis for all animated behaviors. Other sensors are the basis of user interaction, for generating events as the viewer moves through the world or when the user interacts with some input devices. Sensors only generate events. They must be combined with other nodes via *route* statements to have visual effect on the scene.

Scripts and Interpolators

Script nodes can be inserted between event generators (typically sensor nodes) and event receivers. Scripts allow the world creator to define arbitrary behaviors, written in any supported scripting language such as Java and JavaScript. Interpolator nodes are essentially built-in scripts that perform simple animation calculations. They are usually combined with a *TimeSensor* and some nodes in the scene graph to make objects move.

Prototyping: Encapsulation and Reuse

VRML 2.0 includes a prototyping mechanism for encapsulating and reusing a scene graph (the *proto* statement). Geometry, properties, and animations or behaviors can be encapsulated, either separately or together. Prototyping allows the definition of a new node type in terms of a combination of existing node types, which can make VRML easier to use and can reduce the size of VRML files.

Distributed Scenes

VRML 2.0 includes two primitives that allow a single VRML world definition to span the WWW. The *Inline* node allows the inclusion of another VRML file stored anywhere on the Web and the *externproto* statement allows new node definitions to be fetched from anywhere. More generally, *externproto* allows nodes to be defined externally to the VRML file and it is the basic extensibility mechanism for VRML.

2.4.4 Future of VRML Specification

There are several obvious short-term issues for the VRML specification:

- Currently, standard file compression tools are used to compress VRML files, which yields, on average, about a 5:1 compression ratio and requires no extra implementation by the VRML browser. However, many users and developers have requested even more compression. A binary format for VRML could produce much higher compression ratios.

- An External Authoring Interface (EAI) is required. Many users have requested that a standard programmer interface to VRML browsers be defined.

This feature allows technical users to write external programs that communicate with a VRML browser.

2.5 Summary

From the above review, it has been shown that 3D object modeling has a wide spectrum of applications with great potential in the future. However, most existing approaches for 3D object modeling focus only on a certain aspect of either computer vision or computer graphics. They lack the power to bridge the gap between the two fields which were evolving from different research directions decades ago. Despite a number of *ad hoc* techniques, the intrinsic problem of establishing a unified modeling methodology remains unsolved. It is necessary to design a generic representation that supports an integration of a variety of structural models to hold the geometrical, kinematic, physical and behavioral features needed for representing the complexity of 3D objects.

Chapter 3

The Mathematics of AHR

3.1 Introduction

As stated in [70], for any scientific study of modeling, and in particular, for general object modeling, the fundamental problem is: *given a collection of instances of a certain entity, a set of transformations that can be applied on the entity, and an isomorphism or equivalent relation with respect to the transformations, find the general description of the entity and the associated transformations in a mathematical language.* In this chapter, categorical formalism is brought upon to generalize the above problem, since the categorical language, in which there are the essential concepts of objects, transformations, invariance and equivalence, does allow us to express general pattern representation problem in a mathematical formalism.

The universality of the categorical language enables us to define formally some procedures and notions which in the past were empirically defined in computer vision and computer graphics. Many concepts in category theory can play dominant roles in the formalization of most general representation problems. Category theory also allows the establishment of links between superficially very different notions (such as projections and transformations), or between different aspects of the same problem (such as geometrical modeling and physical modeling).

In the following sections, we shall establish a mathematical framework for 3D object modeling with the support of categorical language.

3.2 Category Theory and Graph Based Representation

The theory of categories has been developed for fifty years. It constitutes an autonomous branch of mathematics and has found many applications [76, 70]. In this thesis, the author establishes the attributed hypergraph representation (AHR) for 3D objects based on a categorical framework, and examines it from a mathematical perspective. In order to avoid unnecessary obscurity, a few terms in category theory and graph theory are first explained.

3.2.1 Notations and Definitions

Definitions in Category Theory

Definition 3.1 A *Category* C is given if a class of elements, called *objects*, are given such that:

1. for each pair of objects (O_1, O_2) in C, a map u, called a *morphism* (denoted as $O_1 \xrightarrow{u} O_2$) is given;

2. for each object O in C, there is an *identity* morphism I_O, such that, if $O \xrightarrow{I_O} O'$, then $O' = O$;

3. the *composition* (denoted as $\odot$) of maps of morphisms satisfy:

   (a) if $O_1 \xrightarrow{u} O_2 \xrightarrow{v} O_3$, then $v \odot u$ is the map that maps O_1 to O_2, and then maps O_2 to O_3, denoted as $O_1 \xrightarrow{v \odot u} O_3$;

   (b) if $O_1 \xrightarrow{u} O_2 \xrightarrow{v} O_3 \xrightarrow{w} O_4$, then $O_1 \xrightarrow{v \odot u} O_3 \xrightarrow{w} O_4$ and $O_1 \xrightarrow{u} O_2 \xrightarrow{w \odot v} O_4$;

   (c) identity maps always compose with $O \xrightarrow{u} O'$ to give $I_{O'} \odot u = u$ and $u \odot I_O = u$.

The categories commonly seen in mathematics have objects that are structured sets and morphisms. For example:

- *Set*, consisting of sets and all the maps among sets;
- *Grp*, consisting of groups and the group homomorphisms;
- *Vec*, consisting of vector spaces and the linear transformations;
- *Gph*, consisting of graphs and the graph morphisms. Since graphs can be expressed by *sets*, it is actually a case of *Set*.

Definition 3.2 A morphism u from O to O', denoted as $O \xrightarrow{u} O'$, is called a *retraction* if for each entity in O', by u there is a unique non-null entity in O that corresponds to it; A morphism u from O to O' is called a *coretraction* if for each entity in O, by u there is a unique non-null entity in O' that corresponds to it; A morphism u is an *isomorphism*, if it is both a *retraction* and a *coretraction*, and

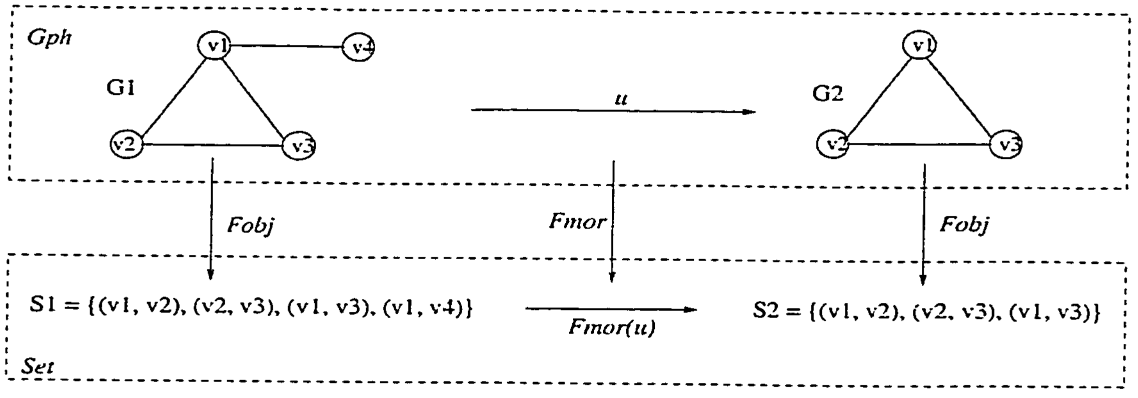

Gph
G1
v1
v2
v3
v4
u
G2
Fobj
Fmor
Fobj
S1 = {(v1, v2), (v2, v3), (v1, v3), (v1, v4)}
Fmor(u)
S2 = {(v1, v2), (v2, v3), (v1, v3)}
Set

Figure 3.1: An example of a functor that maps graph category to set category.

then O is said to be *isomorphic* to O', denoted as $O \sim O'$. If $O \sim O'$ and $O' \sim O''$ then $O \sim O''$.

Definition 3.3 A *covariant functor* F from category C_1 to category C_2 is defined as a pair of maps $F = (F_{obj}, F_{mor})$:

- objects of $C_1 \xrightarrow{F_{obj}}$ objects of C_2
- morphisms of $C_1 \xrightarrow{F_{mor}}$ morphisms of C_2

or simply as $C_1 \xrightarrow{F} C_2$. Usually we refer to a covariant functor as a functor.

Figure 3.1 gives an example of a functor. G_1 and G_2 are two objects in *Gph*, and a morphism u is defined such that $G_1 \xrightarrow{u} G_2$. A covariant functor (F_{obj}, F_{mor}) is defined as the mapping from *Gph* to *Set*. We have $G_1 \xrightarrow{F_{obj}} S_1$ and $G_2 \xrightarrow{F_{obj}} S_2$. Corresponding to u, the morphism between S_1 and S_2 is $F_{mor}(u)$.

Definition 3.4 If a functor F defined on categories C_1 and C_2 preserves all retractions (coretractions, isomorphisms) from C_1 to C_2, it is called a *retraction preserving (coretraction preserving, isomorphism preserving)* functor.

Definition 3.5 The composition of two functors F_1 on categories C_1, C_2 ($C_1 \xrightarrow{F_1} C_2$) and F_2 on categories C_2, C_3 ($C_2 \xrightarrow{F_2} C_3$) is defined as the functor $F_2 \odot F_1$ on C_1 and C_3, such that $C_1 \xrightarrow{F_2 \odot F_1} C_3$.

Definition 3.6 Two categories C_1 and C_2 are called *equivalent* if there exist two functors F_1 and F_2, such that $C_1 \xrightarrow{F_1} C_2$, $C_2 \xrightarrow{F_2} C_1$, and the compositions of the two functors have the properties: (1) $F_2 \odot F_1 = I_{C_1}$; (2) $F_1 \odot F_2 = I_{C_2}$.

Definitions in Graph Theory

Definition 3.7 A *graph* G is an ordered pair (V, E) where $V = \{v_k | 1 \leq k \leq n\}$ is a set of n vertices, and $E = \{e_{ij} | e_{ij} = (v_i, v_j), 1 \leq i \leq n, 1 \leq j \leq n\}$ is a set of edges. Each e_{ij} in E relates two vertices v_i and v_j of V.

Definition 3.8 A graph S is a *subgraph* of G (written as $S \subseteq G$) if $V_S \subseteq V_G$, $E_S \subseteq E_G$. If $S \subseteq G$, G is called a *supergraph* of S.

Definition 3.9 A subgraph S of graph G is said to be *detached* in G if there is no vertex in S adjacent to any vertex of G that is not in S; a non-null graph G is *connected* iff there is no non-null subgraph of G that is detached in G.

Definition 3.10 An *attributed graph* G is a graph defined as an ordered pair $G = (V, E)$ associated with A_V and A_E, where $G = (V, E)$ is a graph, A_V is a finite set of vertex attributes and A_E is a finite set of edge attributes. A_V and A_E's values are defined on continuous domains. Each vertex in V assumes values of attributes from A_V, and each edge in E assumes values of attributes from A_E.

Definition 3.11 A *hypergraph* is an ordered pair $G = (X, Y)$, where $X = \{v_i | 1 \leq i \leq n\}$ is a finite set of vertices and $Y = \{H_j | 1 \leq j \leq m\}$ is a finite set of hyperedges; each H_j is a subset of X such that $H_1 \cup H_2 \cup ..... \cup H_m = X$.

Definition 3.12 An *attributed hypergraph* (AH) G is a hypergraph defined as an ordered pair $G = (X, Y)$ associated with A_X and A_Y, where $G = (X, Y)$ is a hypergraph, A_X is a finite set of vertex attribute and A_Y is a finite set of hyperedge attribute. A_X and A_Y's values are defined on continuous domains Each vertex in X may be assigned values of attribute from A_X, and each hyperedge in Y may be assigned values of attribute from A_Y.

3.2.2 Category Theory and Graph Theory for 3D Object Modeling

Before the 1980's, 2D or 3D objects represented in computer vision and graphics were usually sets of primitives that characterized the geometrical features [64]. With kinematic constraints attached to it, the geometrical feature based representation is able to handle motion analysis and animation if there are only rigid motions involved. Since the 1980's, with the advance of dynamic vision and realistic animation, systems based on physical constraints were developed to solve animation and vision problems in highly deformable situations [25]. The 2D or 3D patterns are characterized by sets of physical parameters, which are used to solve a set of partial differential equations.

Though these representations adequately characterize the corresponding objects at feature levels for computations and manipulations, they lack the capacity to represent more diverse and complex features (such as *behavioral* features) since they lack a symbolic structure. The graph based approaches [71, 90] are designed to solve this problem by mapping the primitive features to vertices and their relations to edges. Symbolic processing on graph representations is possible and a few graph based algorithms are effective for common tasks such as searching, traversing and matching. Since the late 1980's, random graph and attributed hypergraph representations [99, 102, 103] have been developed to provide a generic and unified representation framework.

In the view of category theory, the process of representation is equivalent to applying abstraction functors on a category. The geometrical, physical and graph (or attributed hypergraph) representations mentioned above constitute three different categories:

- the geometrical spaces, with geometrical descriptions and the kinematic transformations defined on the geometrical descriptions, comprise the geometrical category (*Geo*);
- the physical states identified by sets of physical parameters and the state transition functions make up the physical category (*Phy*);
- the graph (or attributed hypergraph) and the graph (or attributed hypergraph) morphisms form the graph category *Gph* (or attributed hypergraph category *Ahp*).

With the concept of functor defined within the scope of computer vision and graphics applications, the three categories can be related as shown in Figure 3.2. The graph based representation can be viewed as the result of structural abstractions from geometrical and physical representations, by applying the functors that map *Geo* and *Phy* to *Gph* respectively. In [99, 102], these abstractions have been

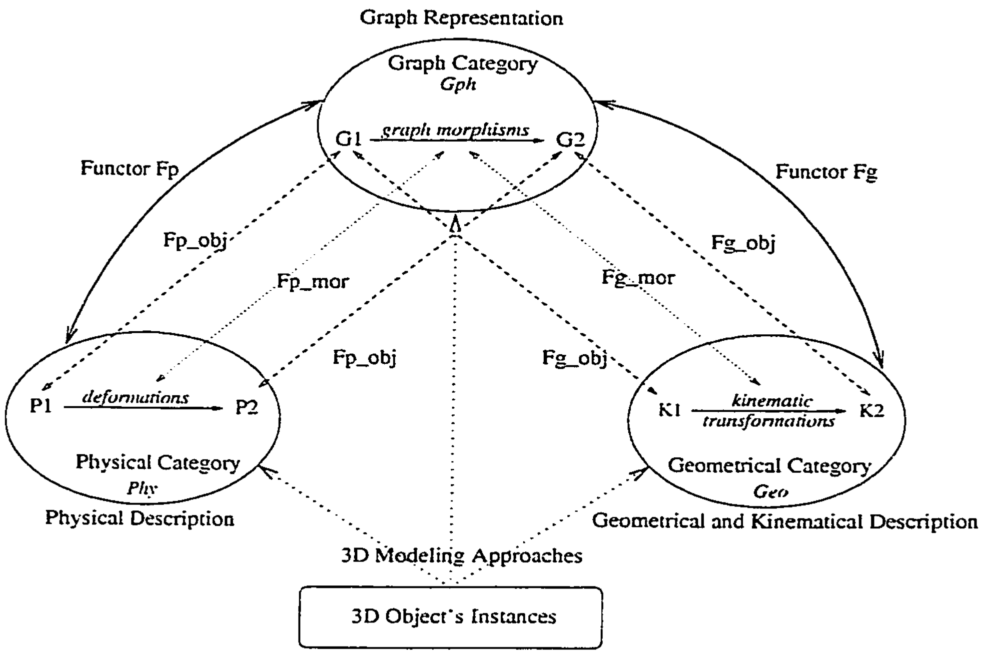


Figure 3.2: The relations between geometrical, physical and graph representations in the view of category theory.

taken by *ad hoc* algorithms. They now can be formally defined and composed by categorical languages.

the following statements summarize the significancies of category theory on 3D object modeling:

- It provides a theoretical background for the abstractions of the classical visual representations into a generic framework: different representations can be labeled by different categories and related by functors.
- It unifies spatial and algebraic approaches in object modeling by manipulating the same notions by the same tools under the same mathematical framework.
- The formalism allows us to create links between superficially different aspects of 3D object modeling.

- Categorical language allows us to present a very general formalization of the computational process in computer vision and graphics: many complicated processes can be explicitly written as the compositions of functors.
- Category theory gives us a mathematical tool to qualitatively evaluate the effectiveness of a representation methodology.

However, due to its generality and universality, category theory cannot alone solve the modeling problems. Since it does not give the methodology to formalize structure, pattern, shape and modeling with categorical language, it is certainly not enough to only facilitate cathegory theory in computer vision and graphics applications. It is necessary to adjoin theories with other mathematical doctrines, which tackle the foundations and the problems in order to define the objects and the transformations in a particular application. In the following sections, the attributed hypergraph representation (AHR) is applied as a solution of 3D object modeling problem, whereas notions in category theory are adopted as the backbones and tools to formalize the AHR.

3.3 Attributed Hypergraph Representation

3.3.1 Dynamic Structure for Attributed Hypergraph

As pointed out in Section 2.1.4, the graph based modeling techniques in the literature have the drawback that their graph structures are rigid. Therefore, they are not appropriate models of deformable objects. In this thesis, hypergraph is implemented with a net-like dynamic data structure (an example is given in Figure 3.3). Such a net based hypergraph is called a *dynamic hypergraph*. The term "dynamic"

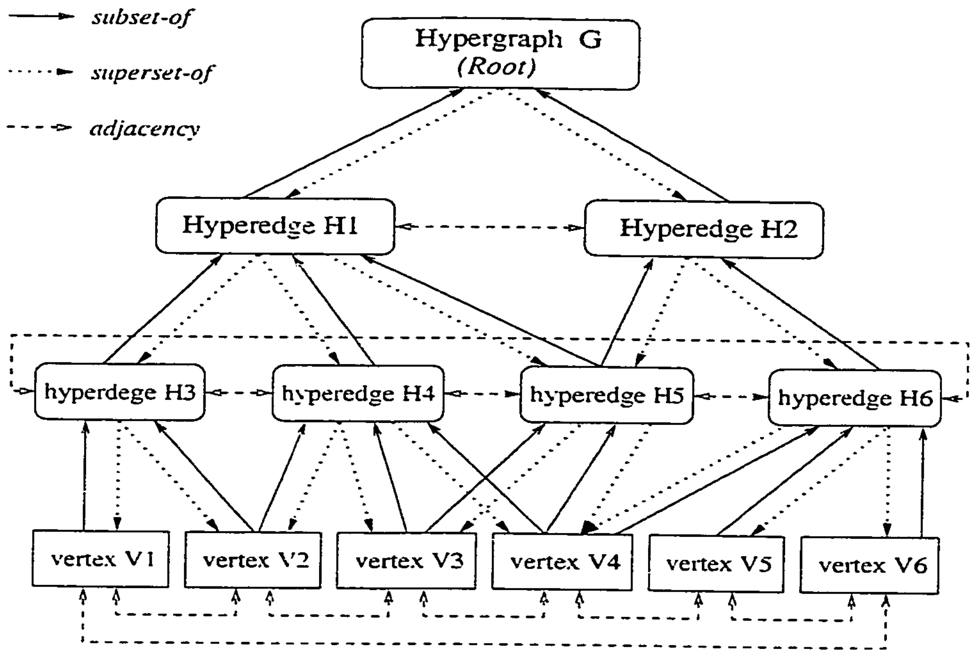


Figure 3.3: The net-like structure for DAH with nodes and links.

signifies that the data structure supports structural operations such as join and subdivision on hypergraphs on the fly, which, in the view of category theory, are related to the morphisms of the object via functors.

It is noticeable that in a hypergraph, a subset of the hypergraph, is also a hypergraph itself. A subset or a superset of a hyperedge is also a hyperedge, and the minimum subset is a vertex. Likewise, a hypergraph or any of its subgraphs can be represented by a collection of hyperedges. This property enables us to define a unified data structure for different types of entities in a hypergraph. In the net-like data structure as illustrated in Figure 3.3, the basic element is called a *node*, shown as a rectangle. The nodes on the bottom layer represent the vertices; the nodes on the intermediate layers are the hyperedges which are supersets of the nodes on their lower layers; the node on the top layer (called the *root*) represents the entire hypergraph.

There are three types of directional links between the nodes. Links from the nodes on the lower layer to the higher layer ones represent the relation "subset-of"; those from the higher (upper) layer nodes to the lower layer ones have the relation "superset-of"; and those between the nodes on the same layer are the adjacency relations. If for each node there is an attribute associated with it, the hypergraph becomes an attributed one. We call this net-like structure a *dynamic attributed hypergraph* (DAH) since both its attribute values and its topology are variable.

With such a data structure, the structural changes on hypergraphs can be handled efficiently. For example, the join of two hyperedges is simply the combination of the two corresponding nodes and the re-organization of the associated links. Graph based computations such as the re-organization of a hyperedge based on certain criteria can be performed by reshuffling the links between the nodes. However, in traditional hypergraph data representations such as the incidence matrix or adjacency matrix, any topological change could initiate a re-organization over many entities. Data elements with or without relations to the nodes to be operated on may have to be accessed.

In a DAH, information represented by nodes on higher layers is abstracted from those on lower layers. The higher layer nodes are important for intelligent manipulation and knowledge processing, while the lower layer ones are crucial for complete and accurate modeling. The nodes on the bottom layer and the links among them construct an *elementary graph*:

Definition 3.13 In a dynamic attributed hypergraph (DAH), the nodes on the bottom layer are called the *elementary nodes*, the links among the elementary nodes are called the *elementary edges*. The attributed graph which consists of the elementary nodes and the elementary edges is called the *elementary graph* of the DAH.

If a DAH has n layers. and the node set on layer i is X_i, then DAH $G = (X, Y)$ can be written in the form of $G = (X_1, X_2, ..., X_n)$ where $X = X_1$ and $Y = \{X_2, ..., X_n\}$. The meanings of the nodes on the intermediate layers normally depend on the applications. For example, in the task of environment modeling for indoor navigation, we can have two intermediate layers on top of the elementary graph:

- the elementary graph characterizes the basic features extracted from the sensory data such as corners and lines;
- the hyperedges on the first intermediate layer represent the organizations of the bottom nodes such that each of them models a piece of flat surface;
- the hyperedges on the second intermediate layer represent higher level knowledge such as the layout of the furnitures in a room;
- the root node represents the entire hypergraph.

In most applications. the elementary graph is very important for a complete representation since it provides the basis to abstract the information required by the nodes on higher layers. In the following sections, the construction of the elementary graph is elaborated.

3.3.2 Triangular Meshes of Range Data

On the bottom layer of a DAH, we need a representation scheme that satisfies the following two requirements simultaneously:

- the ability to represent subtle features for general 3D patterns, from which one can reconstruct a complete object model for computer vision and graphics tasks;
- the suitability to integrate the representation into a structural framework (in particular, to organize the features into the elementary graph of a DAH).

The above requirements arise in many applications, such as medical image processing, museum artifact replication, automatic navigation on natural terrain and reverse engineering of CAD models. The common fact is that usually large amount of raw data are involved, which are necessary to be compiled into more meaningful forms at a higher abstraction level before they become suitable for reasoning or decision-making.

It is ideal to use mesh based representation due to the unstructured nature of general 3D shapes [35]. Since raw data provided by most sensors in 3D applications are actually $2\frac{1}{2}$D (range data) before further processing, a $2\frac{1}{2}$D surface mesh can be used at the initial stages to approximate the shape and to compress data before a complete 3D model is synthesized.

In this thesis, a triangular mesh modeling and coarsening algorithm is adopted from [35] as the base representation. It is justifiable to use triangular mesh, a special case of polygonal mesh, for the following reasons:

- it can be easily generated from various sensory data types, such as data from stereo CCD cameras, sonar sensors or laser range finders;
- it can achieve multi-resolution modeling, where the highest precision could be as high as what the sensory device can achieve;
- it can be converted into other data formats with very little information loss;

- it consists of vertices and edges, therefore, it is straightforward to be reorganized into a graph structure.

Compared with other mesh based methodologies such as NURBS, a triangular mesh has the advantages that it is simple and flexible. If a common representation is to be integrated into a unified framework from different features (such as geometrical features and physical features), a flexible representation should be adopted and pursued. The traditional concerns about its accuracy, storage and combinatorics are not really problems on today's computers. Many operations on triangular meshes are faster and require less storage than comparable NURBS solutions in some commercial packages [12].

A triangular mesh $\mathcal{T}$ representing range data must satisfy the following requirements [14]:

- $\mathcal{T}$ should be topologically and geometrically correct;
- the quality of $\mathcal{T}$ should be as high as possible and contain as few badly shaped triangles as possible;
- the vertices of $\mathcal{T}$ should be positioned on the surface of the object:
- $\mathcal{T}$ should be *boundary conforming*: triangles should form a full tessellation of the object's surface.

With the above constraints, suppose that two given sets E_f and V_f define a triangulation which embeds edges in E_f and vertices in V_f, it can be shown that, for a triangulation of E_f and V_f, the number of the triangles is $N_t = 2N_v - N_b - 2$ and the number of the edges is $N_e = 3N_v - N_b - 3$, where N_v is the cardinality of V_f and N_b is the number of points on the convex hull of V_f [57]. Typically,

$N_b = O(\sqrt{N_v})$. This produces approximately $3N_v$ edges and $2N_v$ triangles. In any case. the number of edges or triangles is each $O(N_v)$.

Triangular Mesh Approximation of Object Surfaces

An initial mesh is constructed in two stages. First, a feature detection phase identifies the important characteristics of the 3D scene. The second stage triangulates the resulting features and forms the mesh.

The feature detection can be done in several ways. Most techniques use slope and curvature estimations on the range data to extract geometrical features. Such implementations, however, are costly: they require integer approximations to reduce the complexity of floating point computations, or use specialized hardware to reduce the execution time. Though some work has been done for the automatic geometrical feature extraction from range data [27, 77, 83], many of the techniques apply only to simple examples without explicitly handling the discontinuities occurred due to the limitations of the range sensors. In this thesis, the algorithm adopted was proposed in [35], in which, however, the surface features are detected from the original triangulation of the range image. As an enhancement over the original algorithm, the type of the features to be extracted is generalized not only for geometrical ones, but also for physical or appearance ones. Usually the features are not evenly distributed across the region, which further constitutes higher order information that is important for symbolic processing.

The second stage consists of the construction of the $2\frac{1}{2}$D triangular mesh that embeds the vertices and the edges detected during the first stage. Among all possible triangulations of a set of edges and vertices. the Delaunay triangulation is most commonly used due to its provable properties [38, 39, 78, 42]. Software implementations [40] can be obtained from the public domain library *Netlib*. This implementation uses a sweep-line technique that triangulates n 2D points in $O(n \log n)$ time with $O(n)$ space.

Since this initial Delaunay triangulation optimizes only the properties in the planar projection of the data, it does not necessarily conform to the characteristics of the actual surface or the actual type of the features that we select. Therefore, the raw mesh is improved with the guidance of following features:

- range height for geometrical features;
- local color or texture vectors for appearance features;
- local mass density or elastic force for physical features.

Feature Based Mesh Coarsening

The initial triangular mesh obtained from the raw range data is normally too dense to perform further analysis required in most applications, such as scene understanding and path planning. The algorithm adopted from [35] for feature based mesh coarsening is briefly described in Appendix A. It initially uses a *thick representation* for the triangulation. Its data structure T is assumed to include:

- the list of vertices, their 3D coordinates and their associated feature values;
- the list of vertex indices depicting their incident edges;
- the list of triangle indices with the vertices defining them;
- the list of edge indices with the vertices incident to them.

Ultimately, the output would be the set of selected significant vertices together with a set of edges, approximating the connected components in the original mesh.

Attributed Hypergraph Based on Triangular Mesh

As described in [34], an object can be approximated by triangular meshes induced by salient geometrical features of the object's surfaces. The structure of the mesh fits in excellently graph based representations. Hence, it is adopted as the base of AHR in this thesis. In the vertex-edge structure, a triangular mesh $\mathcal{T}$ can be written in the form of $\mathcal{T} = (V_t, E_t)$ where V_t is the vertex set and E_t is the edge set of the mesh. A full representation of the mesh can assume the form of an attributed graph, which then constructs the base of the AHR.

Definition 3.14 A representative attributed graph (RAG) $G = (V, E)$ of a triangular mesh $\mathcal{T}$ is an attributed graph constructed in the following manner:

1. for each vertex $v_t \in V_t$, there is a vertex $v_a \in V$ corresponding to it;
2. for each edge $e_t \in E_t$, there is an edge $e_a \in E$ corresponding to it;
3. v_t's features are mapped to v_a's features;
4. e_t's features are mapped to e_a's features.

The AHR of a triangular mesh, which represents a 3D shape, is based on such RAG. In the net-like DAH data structure (refer to Figure 3.3), the elementary nodes of a DAH (i.e., the nodes on the bottom layer) consist of the vertices in the RAG, and the elementary edges are copied from the edges in the RAG. The surface's properties attached to the mesh, such as location, orientation, area, color, texture, mass density or elasticity, are mapped to the attributes associated with the vertices or edges. Thus, the RAG of the mesh directly constitutes the elementary graph of the AHR.

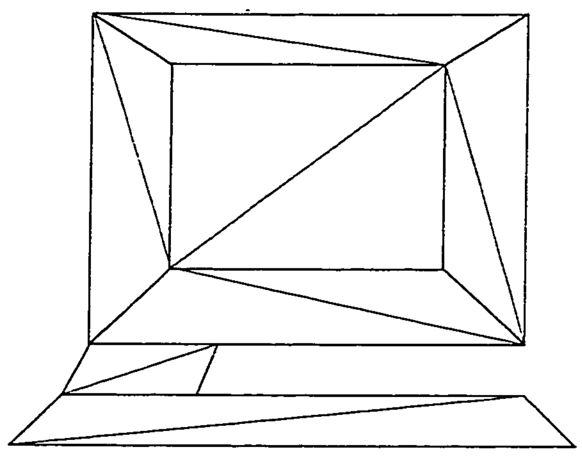
The hyperedges in the DAH are generated based on the selected attribute types and values of their elements that correspond to certain key features:

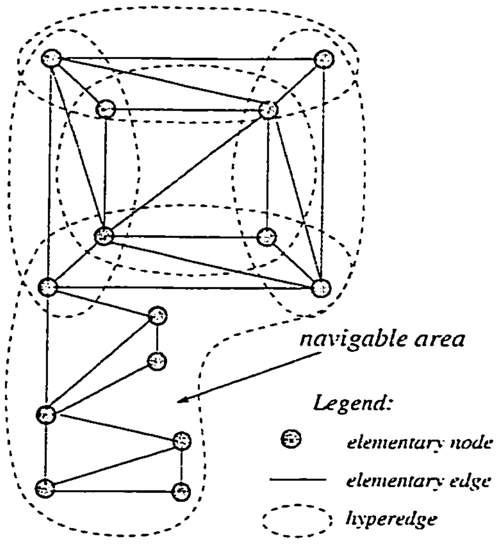
Definition 3.15 Suppose that $X = \{x_1, x_2, ..., x_n\}$ is the vertex set of hypergraph $G = (X, Y)$, $E = \{x_{e_1}, x_{e_2}, ..., x_{e_m}\}$ ($E \subseteq X$, $e_k < n, k = 1, 2, ..., m$) is a hyperedge, and $\{a_{e_1}\}, \{a_{e_2}\}, ..., \{a_{e_m}\}$ are the vertex attribute sets associated with vertices $x_{e_1}, x_{e_2}, ..., x_{e_m}$ respectively. We say that E is a hyperedge induced by attribute value a, if for the selected attribute value a, we have:

1. $a \in \{a_k\}$ if the corresponding vertex $x_k \in E$;
2. $a \notin \{a_k\}$ if the corresponding vertex $x_k \notin E$;
3. E is connected.

Different from edges in a classical attributed hypergraph, hyperedges in a DAH can be generated from the topologies and/or the attributes of the nodes on any of their lower layers. The elements of X in E in the above definition can be the vertices on the bottom layer of a DAH, or the nodes on any of the layers below the hyperedge layer. In a triangular mesh based DAH, each element in a hyperedge can signify a triangle in the RAG, and the hyperedges represent the organizations of the triangles based on certain constraints.

For example, in robot vision related applications, we need to know the navigable area in a 3D scene. The features to establish a hypergraph organization are the orientations and the connectivities of the triangles. Triangles that are connected with each other and with the differences of their normal vectors not exceeding a preset threshold can be grouped together to form a surface patch. The representation of a surface patch in DAH is a hyperedge node at the layer above the nodes that stand



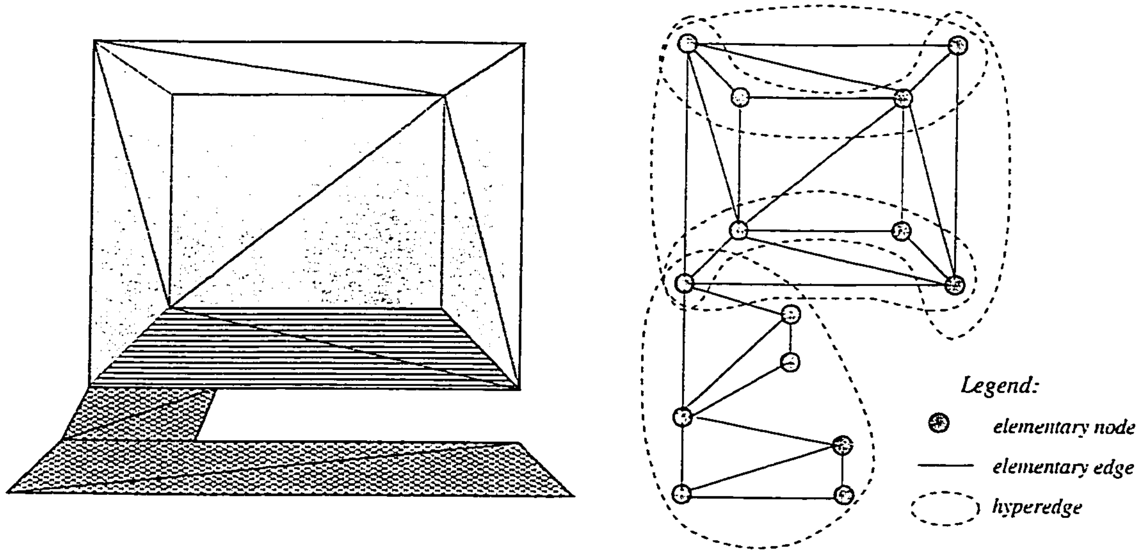


(a) (b)

Figure 3.4: (a) A sample indoor 3D scene; (b) The AHR for vision guided navigation.

for triangles. Figure 3.4 (a) shows a sample 3D scene with simple triangles as its surfaces. The AHR generated for navigation guidance in a 3D scene is illustrated in Figure 3.4 (b). For simplicity, it only shows the elementary graph and the hyperedges (illustrated by dashed lines) that comprise of collections of triangles. The navigable area is defined by a hyperedge which contains the floor area.

Another example of hyperedge construction is in the task of texture binding for *augmented reality* (refer to Chapter 4). As stated in Section 4.5.2, to form an augmented reality from real scenes and virtual elements, natural looking textures have to be extracted from 2D views and then transposed to 3D surfaces. In this task domain, the features to construct the hyperedges are the surface textures and the connectivity properties of the triangles. Figure 3.5 (a) shows a similar 3D scene as the one in Figure 3.4 but with textures mapped to the triangles. Figure 3.5 (b) shows the corresponding AHR for texture binding. The hyperedges are different



(a) (b)

Figure 3.5: (a) A sample indoor 3D scene with textured surfaces; (b) The AHR for texture binding in augmented reality.

from those in Figure 3.4 (b). With the AHR, we can change the wall paper of the room by simply changing the attribute value of the corresponding hyperedge.

More details about attributed hyperedge generation will be discussed in Chapters 4 and 5.

3.3.3 Primary Operators on Hypergraphs

Corresponding to the motions of a 3D pattern, the transformations on its AHR are represented by a series of operators. In this section, the definitions of the primary operators on attributed hypergraph are provided. In the applications presented in Chapters 4 and 5, these operators play an important role for 3D pattern analysis and manipulation.

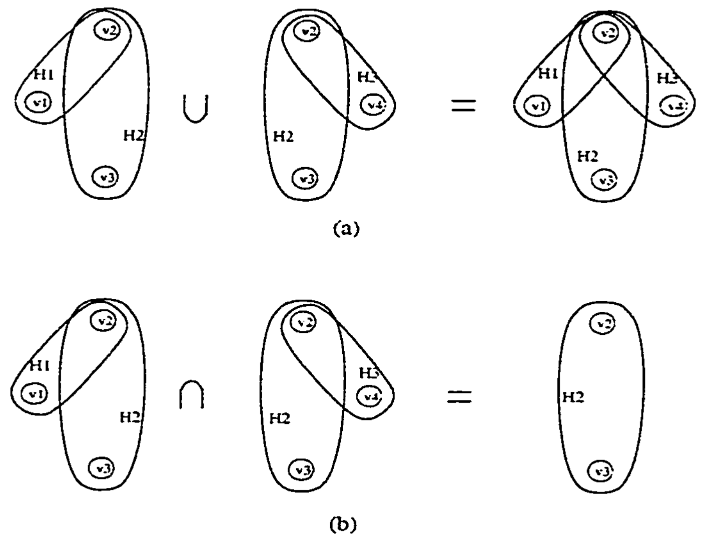


Figure 3.6: The primary operators for attributed hypergraphs: (a) *union* and (b) *intersection*.

Definition 3.16 The *union* of two attributed hypergraphs $G_1 = (X_1, Y_1)$ and $G_2 = (X_2, Y_2)$, is an attributed hypergraph $G = (X, Y)$, denoted as $G = G_1 \bigcup G_2$, such that $X = X_1 \cup X_2$ and $Y = Y_1 \cup Y_2$. All vertices and hyperedges in G preserve their attribute values.

Definition 3.17 The *intersection* of two attributed hypergraphs $G_1 = (X_1, Y_1)$ and $G_2 = (X_2, Y_2)$, is an attributed hypergraph $G = (X, Y)$, denoted as $G = G_1 \bigcap G_2$, such that $X = X_1 \cap X_2$ and $Y = Y_1 \cap Y_2$. All vertices and hyperedges in G preserve their attribute values.

Figure 3.6 illustrates the operations of *union* and *intersection* on two simple hypergraphs.

Definition 3.18 The *dichotomy* of a vertex v_d in attributed hypergraph $G = (X, Y)$, denoted as $\ominus(v_d) = (v_{d_1}, v_{d_2})$, is obtained by taking the following steps:

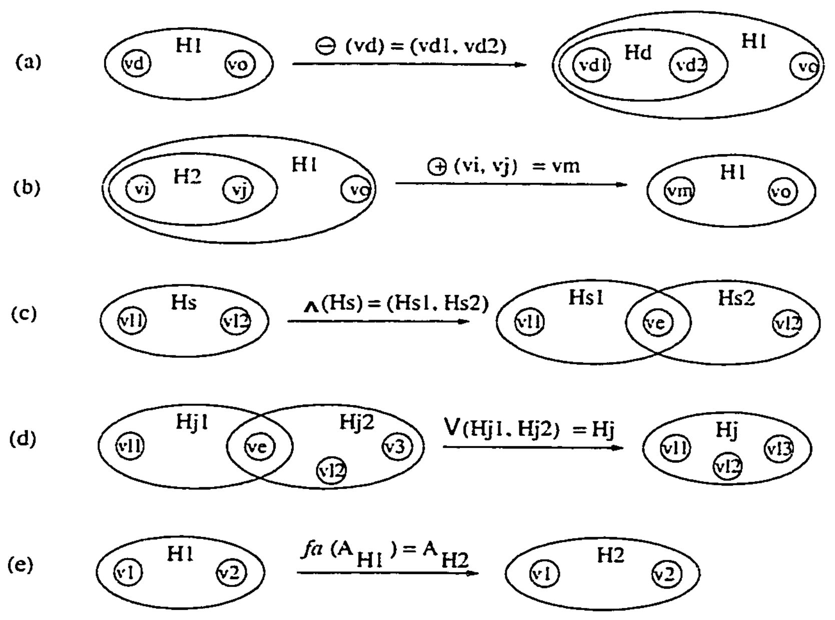


Figure 3.7: The primary operators for an attributed hypergraph: (a) *dichotomy*; (b) *merge*; (c) *subdivision*; (d) *join*; (e) *attribute transition*.

1. add two vertices v_{d_1}, v_{d_2} to X, which assume the attribute value from v_d;

2. replace v_d in all hyperedges by v_{d_1} and v_{d_2};

3. add a new hyperedge H_d to Y where $H_d = \{v_{d_1}, v_{d_2}\}$, H_d's attribute value is the average attribute value of all hyperedges that contain v_d before the operation;

4. remove v_d from X.

An example of *dichotomy* is shown in Figure 3.7 (a).

Definition 3.19 If in attributed hypergraph $G = (X, Y)$, two vertices v_i and v_j are adjacent in n hyperedges H_{l_1}, H_{l_2}, ..., H_{l_n} ($n \geq 1$), then the *merge* of v_i and v_j, denoted as $\oplus(v_i, v_j) = v_m$, is obtained by the following steps:

1. add a vertex v_m to X, v_m's attribute value is the average of those of v_i's and v_j's;
2. remove v_i and v_j from X;
3. in $H_{l_1}, H_{l_2}, ..., H_{l_n}$, replace v_i **and** v_j by a single vertex v_m;
4. replace v_i **or** v_j by v_m in any other hyperedges in Y;
5. if a hyperedge contains only v_i or v_j, it is removed.

An example of *merge* is shown in Figure 3.7 (b).

Definition 3.20 The *subdivision* of hyperedge $H_s = \{v_{l_1}, v_{l_2}, ..., v_{l_n}\}$ in attributed hypergraph $G = (X, Y)$ (where $v_{l_i} \in X$, $1 \leq i \leq n$, $H_s \in Y$), denoted as $\bigwedge(H_s) = (H_{s_1}, H_{s_2}, ..., H_{s_n})$, is obtained by the following steps:

1. add a new vertex v_e to X, v_e's attribute value is the average attribute value of vertices $v_{l_1}, v_{l_2}, ..., v_{l_n}$;
2. add n hyperedges $H_{s_1}, H_{s_2}, ..., H_{s_n}$ to Y, such that $H_{s_i} = \{v_{l_i}, v_e\}$ $(1 \leq i \leq n)$, each of which assumes the attribute value from H_s;
3. remove H_s from Y.

Figure 3.7 (c) illustrates an example of *subdivision* on a simple hypergraph.

Definition 3.21 If in attributed hypergraph $G = (X, Y)$, H_{j_1} and H_{j_2} are two hyperedges with at least one common vertex, then the *join* of H_{j_1} and H_{j_2}, denoted as $\bigvee(H_{j_1}, H_{j_2}) = H_j$, is to:

1. add a new hyperedge H_j to Y such that $H_j = H_{j_1} \cup H_{j_2}$. H_j's attribute value is the average of those of H_{j_1}'s and H_{j_2}'s;
2. in H_j, remove the vertices that are common in H_{j_1} and H_{j_2};
3. remove H_{j_1} and H_{j_2} from Y.

An example of *join* is shown in Figure 3.7 (d).

Definition 3.22 The *attribute transition* of a hyperedge $H \in Y$ (or vertex $v \in X$) in an attributed hypergraph $G = (X, Y)$, denoted as $A'_H = f_a(A_H)$ (or $A'_v = f_a(A_v)$), is a mapping $A_H \xrightarrow{f_a} A'_H$ (or $A_v \xrightarrow{f_a} A'_v$) which transforms the attribute value A_H (or A_v) without any change to the connectivities (refer to Figure 3.7 (e)).

With the net-like structure of an attributed hypergraph, in many cases, the primary operations can be performed with the complexity of $O(1)$: only the objective nodes and their ancestors are required to be visited. However, we have to consider the worst case. For example, a subdivision on a vertex at the bottom layer will produce a new node. The emerging of the new node may force an attribute change of its neighbors. Furthermore, the attribute changes may have to propagate to other nodes that are not in the same layer. In the worst case, the changes will propagate to the top layer. In a DAH with n nodes, there are at most $n(n-1)$ bi-relations among all the nodes across all layers. Since the attributes to be updated for all nodes can be performed with $O(n)$ and each update is at most spread out with $n(n-1)$ links, the operation cost in the worst case is $O(n^3)$.

Definition 3.23 Suppose that G_1, G_2, G_3 and G_4 are four attributed hypergraphs, and op_a and op_b are two primary operators applied on entities (or entity sets) H_a

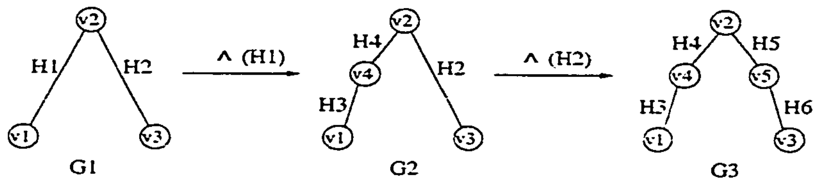


Figure 3.8: A compound operator defined on a simple graph.

of G_1 and H_b of G_3 respectively, such that $G_1 \overset{op_a(H_a)}{\longrightarrow} G_2$ and $G_3 \overset{op_b(H_b)}{\longrightarrow} G_4$. The *composition* of op_a and op_b, denoted as $\copyright(H_a, H_b) = op_b(H_b) \odot op_a(H_a)$, which gives $G_1 \overset{\copyright(H_a,H_b)}{\longrightarrow} G_4$, is defined iff $G_2 = G_3$.

Figure 3.8 shows an example. The operators in the figures give $G_1 \overset{\bigwedge(H_1)}{\longrightarrow} G_2$ and $G_2 \overset{\bigwedge(H_2)}{\longrightarrow} G_3$. With a composition operator $\copyright$ applied on H_1 and H_2 and defined as $\bigwedge(H_2) \odot \bigwedge(H_1)$, it can be written as $G_1 \overset{\copyright(H_1,H_2)}{\longrightarrow} G_3$

In the view of category theory, the primary operators and their compositions constitute the morphisms between different attributed hypergraphs in the AH category *Ahp*. Through the mapping of functors, they correspond to the motions of the 3D patterns.

We shall now formally define the morphisms among attributed hypergraphs to construct the category *Ahp*. It is noticed that given two attributed hypergraphs G_i and G_j, the series of primary operators which maps G_i to G_j may not be unique. Nevertheless, similar to the definition of morphisms for *Set* given in [18], we can define the map between G_i and G_j as the set of *all* operator sets that take G_i as the input and output G_j, denoted as S_{ij}. Defining the objects as the attributed hypergraphs and the morphism between two objects G_i and G_j as a triplet $u_{ij} = (G_i, S_{ij}, G_j)$, we prove the following theorem:

Theorem 3.1 The attributed hypergraphs and the AH operators form a category *Ahp*.

[Proof]

Proof is made by the definition of category given in Definition 3.1.

Given two attributed hypergraphs G_i and G_j, the u_{ij} between them is admissible since it is defined as a triplet (G_i, S_{ij}, G_j). For two morphisms $u_{ij} = (G_i, S_{ij}, G_j)$ and $u_{kl} = (G_k, S_{kl}, G_l)$, $u_{ij} = u_{kl}$ iff $G_i = G_k$, $G_j = G_l$ and $S_{ij} = S_{kl}$.

For each object G in *Ahp*, there is a unique identity morphism $I_G = (G, S_I, G)$ where S_I is the unique identity attribute transition operator $f_{a_I}(G) = G$.

Similar to Definition 3.23, the composition $u_{ij} \odot u_{lm}$ of the morphisms $u_{lm} = (G_l, S_{lm}, G_m)$ and $u_{ij} = (G_i, S_{ij}, G_j)$ is defined iff $G_m = G_i$. In such case, $u_{ij} \odot u_{lm}$ has G_l as the input and takes value of G_j, i.e., $G_l \xrightarrow{u_{ij} \odot u_{lm}} G_j$.

From the above composition of operators, it is immediate that: for attributed hypergraph G_i, G_j, G_k and G_l, if $G_i \xrightarrow{u_{ij}} G_j$, $G_j \xrightarrow{u_{jk}} G_k$ and $G_k \xrightarrow{u_{kl}} G_l$, compositions $u_{jk} \odot u_{ij}$ and $u_{kl} \odot u_{jk}$ can be defined. By the property of the compositions, we have $G_i \xrightarrow{u_{jk} \odot u_{ij}} G_k \xrightarrow{u_{kl}} G_l$ and $G_i \xrightarrow{u_{ij}} G_j \xrightarrow{u_{kl} \odot u_{jk}} G_l$.

As the indentity map for attributed hypergraph is the indentity attribute transition operator f_{a_I}, it does not change the attributed hypergraph's topology, nor its attributes. Therefore, it always composes with u_{ij} to give $I_{G_j} \odot u_{ij} = u_{ij}$, $u_{ij} \odot I_{G_i} = u_{ij}$.

By showing that *Ahp* has all the properties of a category, we demonstrate that attributed hypergraphs associated with their operators form a category.

□

3.3.4 3D Object Modeling Using Attributed Hypergraph and Attributed Hypergraph Operators

In the context of this thesis research, three levels of information are involved in the AHR based modeling methodology:

- *realization level:*

  This is the visual level depicting the actual scene. At this level, 3D objects are represented by graphical elements. It serves as a platform on which the reconstructed 3D scene is rendered and back-projected.

- *feature level:*

  At this level we have the extracted features (geometrical, kinematic or physical) from the 3D scenes, or the synthesized features from the object models. The features at this level can be used to extract symbolic information, or to reconstruct 3D scenes through the back-projection to the realization level.

- *symbolic level:*

  This is the representation level where symbolic information is abstracted and then organized in an attributed hypergraph. The manipulations are mainly performed at this level.

Figure 3.9 illustrates the relations among the information on the three levels with a simple example. We can see that the use of AHR enables us to manipulate the 3D data focusing only on the selected feature types on a relatively high level. Thus, the amount of information to be processed is reduced significantly. At the representation level instead of the realization level or feature level, we can also perform transformations in the form of AH operators (refer to Section 3.3.3) on

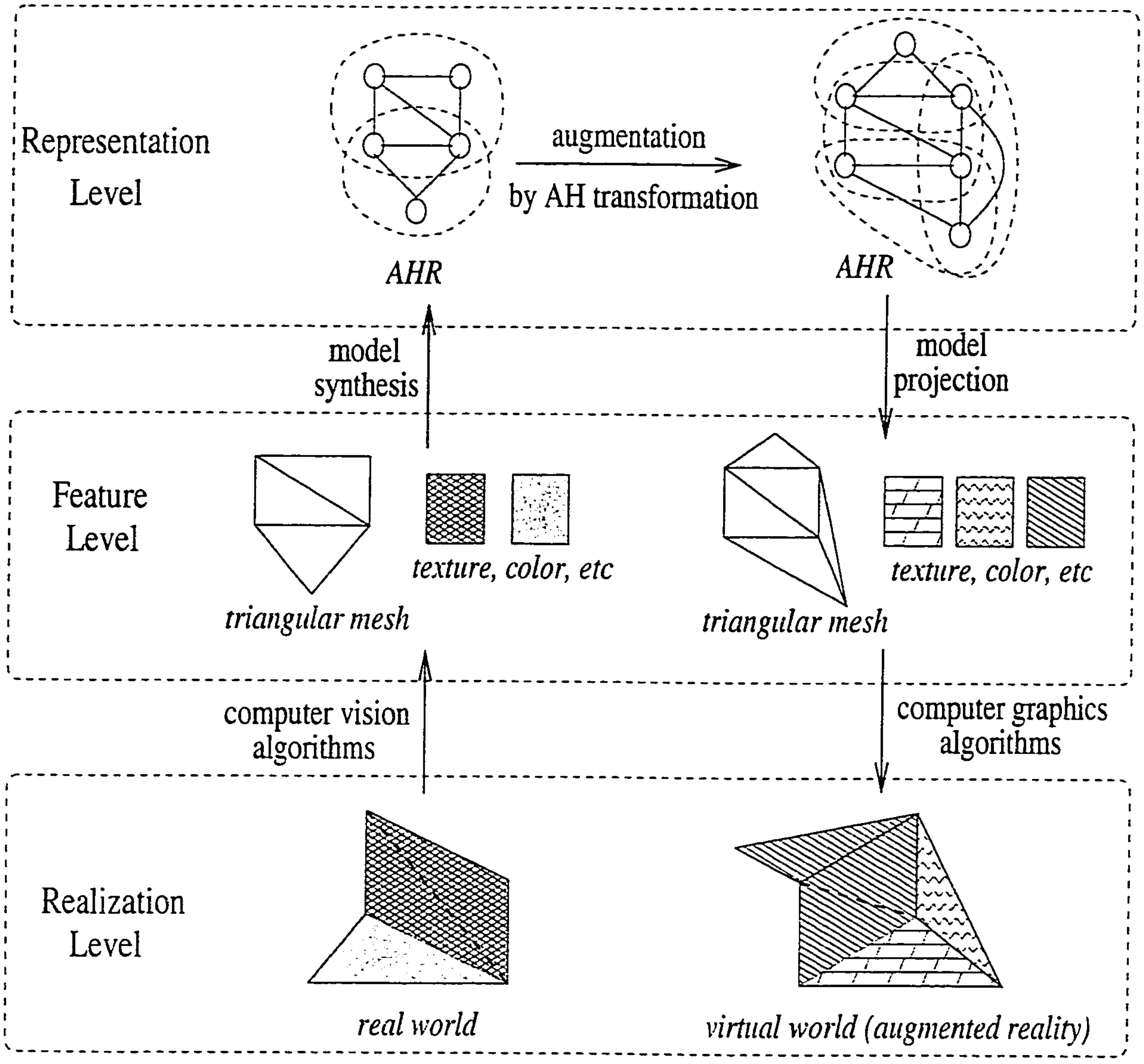


Figure 3.9: The relations among realization level, feature level and representation level.

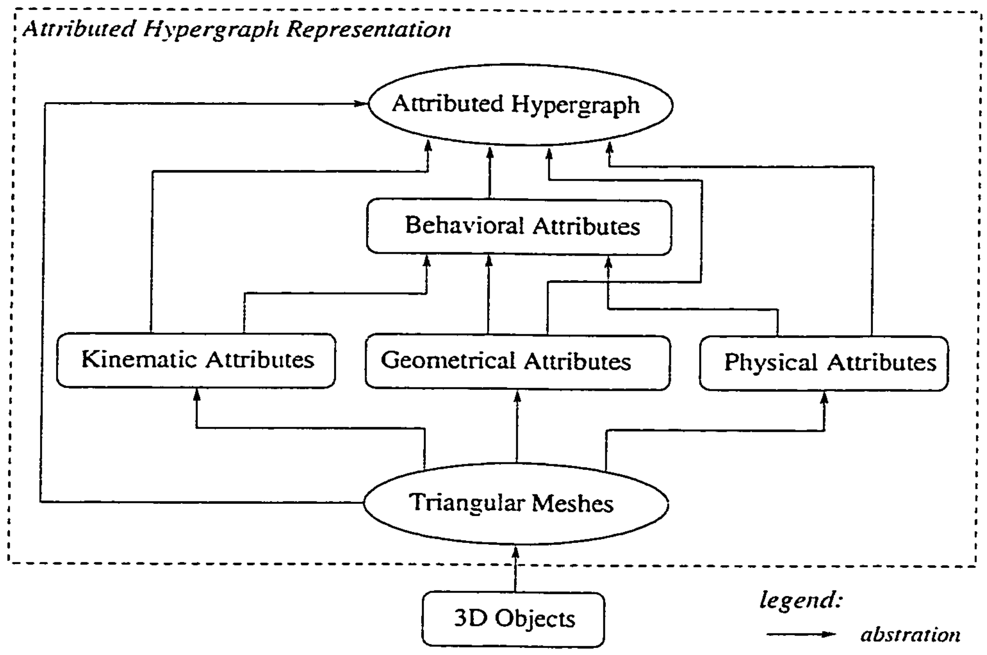


Figure 3.10: Geometrical, kinematic, physical and behavioral features and their relations in AHR.

the attributed hypergraph to represent various changes of the scene, such as scene augmentation and 3D morphing.

When transformations are applied over an AHR, kinematic features and physical features are sufficient to characterize the rigid or non-rigid motions of the object. Based on the kinematic and physical attributes, we further introduce the behavioral attributes, a high-level feature type to model constraints, ego-motions and other intelligent behaviors of the objects. As shown in Figure 3.10, the kinematic and physical attributes are extracted directly from the triangular mesh, while the behavioral attributes are abstracted from the kinematic and physical attributes. The attributed hypergraph is constructed from the triangular mesh, together with all three types of attributes.

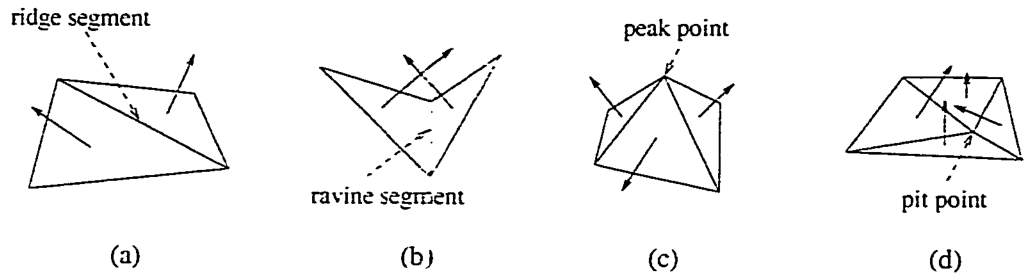


Figure 3.11: Geometrical features in triangular meshes: (a) ridge line segment; (b) ravine line segment; (c) peak point: (d) pit point.

In the following, we provide the details of the 3D object modeling methodology with integrated geometrical, kinematic, physical and behavioral features.

Modeling of Geometrical Shapes

We have adopted the triangular mesh as the base of the AHR. Since the meshes were originally introduced for shape approximations, modeling of geometrical shapes with AHR is straightforward:

A. *Realization level:*

The surfaces of a 3D object are approximated by different patches in triangular meshes: each patch itself constitutes a mesh with $2\frac{1}{2}$ range data. For rendering, they can be back-projected onto the viewing plane in mesh form or in a surface form after being fitted by the surface spline algorithm described in [30].

B. *Feature level:*

At the feature level, relevant information is the geometrical features extracted directly from the meshes. The detection of the geometrical features is straightforward: it suffices to study neighboring triangles or neighboring patches of triangles, and the common edges/boundaries among them. to determine the low level shape features such as feature edges and corners. and high level shape features such as *ridges*,

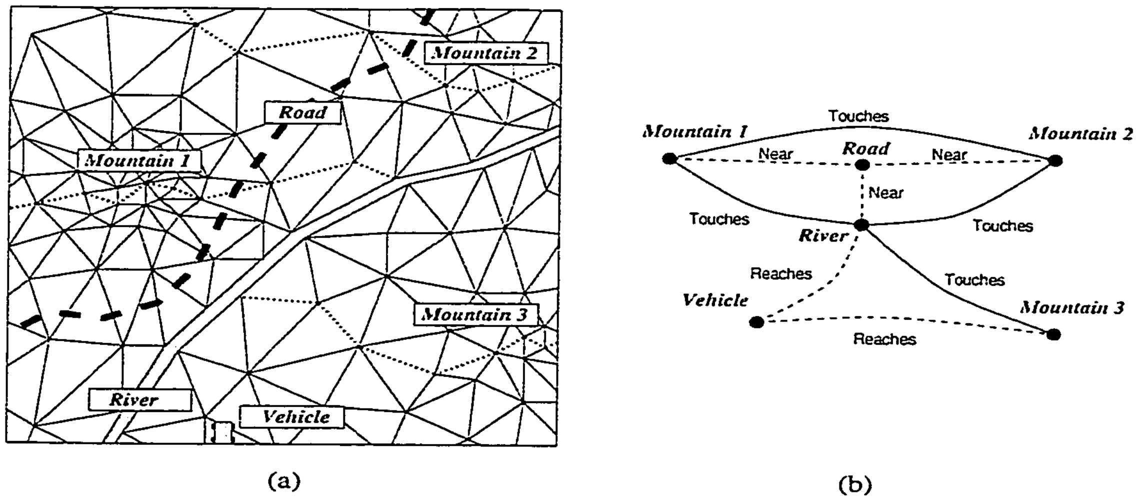


Figure 3.12: (a) A sample outdoor scene; (b) the corresponding conceptual graph.

ravines, *peaks*, *pits* and *saddles* (refer to Figure 3.11).

C. *Symbolic level:*

Task dependent symbolic information about 3D shapes can be extracted. For example, for vision guided navigation purpose, such symbolic information may include a conceptual map built upon the ridges. ravines, peaks, pits and saddles at the feature level [35]. Figure 3.12 (copied from [35]) illustrates an example of the conceptual map of a natural terrain.

We shall now discuss the possibility of organizing triangular meshes and the associated kinematic transformations. which together compose the geometrical and kinematic description, into a *category*. The terminologies introduced in Section 3.2.1 are referred.

Suppose that the objects of category *Geo* are n triangular meshes $\mathcal{T}_1$, $\mathcal{T}_2$, ..., $\mathcal{T}_n$, each of which approximates a 3D shape. $\mathcal{T}_i$ $(1 \leq i \leq n)$ in *Geo* is represented by its RAG, denoted as T_i. Let the transformation associated with T_i and T_j be represented by a triplet $u_{ij} = (T_i, d_{ij}, T_j)$ where d_{ij} includes collection of all the operator sets that transform RAG T_i to RAG T_j, then we prove the following theorem:

Theorem 3.2 The triangular mesh based geometrical descriptions of 3D patterns in the form of RAG's with the associated transformations form a category *Geo*.

[Proof]

u_{ij} is admissible since it is defined as a triplet (T_i, d_{ij}, T_j). For two morphisms $u_{ij} = (T_i, d_{ij}, T_j)$ and $u_{kl} = (T_k, d_{kl}, T_l)$, $d_{ij} = d_{kl}$ does not necessarily mean that $u_{ij} = u_{kl}$. One has $u_{ij} = u_{kl}$ iff $T_i = T_k$, $T_j = T_l$ and $d_{ij} = d_{kl}$.

For each object T in *Geo*, there is a unique identity morphism $I_T = (T, d_I, T)$ where d_I is the identity attribute transition $f_{a_I}(T) = T$.

Similar to Definition 3.23, the composition $u_{ij} \odot u_{lm}$ of the morphisms $u_{lm} = (T_l, d_{lm}, T_m)$ and $u_{ij} = (T_i, d_{ij}, T_j)$ is defined iff $T_m = T_i$. If $T_m = T_i$, then $u_{ij} \odot u_{lm}$ takes T_l as the input and has output of T_j, i.e., $T_l \xrightarrow{u_{ij} \odot u_{lm}} T_j$.

From the above construction of compound operators, it is immediately seen that: for RAG's T_i, T_j, T_k and T_l, if $T_i \xrightarrow{u_{ij}} T_j$, $T_j \xrightarrow{u_{jk}} T_k$ and $T_k \xrightarrow{u_{kl}} T_l$, compositions $u_{jk} \odot u_{ij}$ and $u_{kl} \odot u_{jk}$ can be constructed. By the composition of AH operators defined in Definition 3.23, we have $T_i \xrightarrow{u_{jk} \odot u_{ij}} T_k \xrightarrow{u_{kl}} T_l$ and $T_i \xrightarrow{u_{ij}} T_j \xrightarrow{u_{kl} \odot u_{jk}} T_l$.

It is also obvious that the identity maps always compose with u_{ij} to give $I_{T_j} \odot u_{ij} = u_{ij}$ and $u_{ij} \odot I_{T_i} = u_{ij}$.

By showing that *Geo* has all the properties defined in Definition 3.1, we demonstrate that the geometrical descriptions of 3D patterns together with the associated transformations form a category.

□

Introducing the definition of the functor $F_g = (F_{g-obj}, F_{g-mor})$ which maps the geometrical descriptions in *Geo* to entities of AHR in AH category *Ahp*, we can express the formation of a DAH from the geometrical description by F_{g-obj} mathematically. Suppose that a geometrical description based on a triangular mesh $\mathcal{T} = \{t_i | i = 1, 2, ..., n\}$ is in the form of its corresponding RAG $G = (V_t, E_t)$, the DAH is $H = (X, H_{base}, H_t, H_{hyper})$, where X is the vertex set, and H_{base}, H_t and H_{hyper} are the edge/hyperedge sets on higher layers respectively. We have:

1. elementary node mapping:

$$v_t \xrightarrow{F_{g-obj}} v_g \tag{3.1}$$

   where $v_t \in V_t$ and $v_g \in X$, both with attribute value $a_v = \{x_t, y_t, z_t\}$ denoting the 3D position of v_t on the object's surface.

2. elementary edge mapping:

$$e_t = \{v_{t_1}, v_{t_2}\} \xrightarrow{F_{g-obj}} h_{base}\{v_{g_1}, v_{g_2}\} \tag{3.2}$$

   where $e_t \in E_t$, $h_{base} \in H_{base}$, such that by Equation 3.1, $v_{t_i} \xrightarrow{F_{g-obj}} v_{g_i}$ $(i = 1, 2)$; e_t and h_{base} both have attribute value a_e denoting the directional vector for the edge e_t.

3. triangle mapping:

$$t_t = \{v_{t_1}, v_{t_2}, v_{t_3}\} \xrightarrow{F_{g-obj}} h_t = \{v_{g_1}, v_{g_2}, v_{g_3}\} \tag{3.3}$$

   where t_t is a triangle in $\mathcal{T}$ with vertices v_{t_1}, v_{t_2}, v_{t_3}, $h_t \in H_t$ is a hyperedge with vertices v_{g_1}, v_{g_2}, v_{g_3}, such that by Equation 3.1, $v_{t_i} \xrightarrow{F_{g-obj}} v_{g_i}$ $(1 \le i \le 3)$. t_t and h_t have attribute value $a_t = \vec{t_i}$ denoting the normal vector of t_i.

4. hyperedge mapping:

$$S_t = \{t_{t_i} | 1 \leq i \leq k\} \overset{F_{g-obj}}{\longrightarrow} h_{hyper} = \{h_{t_i} | 1 \leq i \leq k\} \tag{3.4}$$

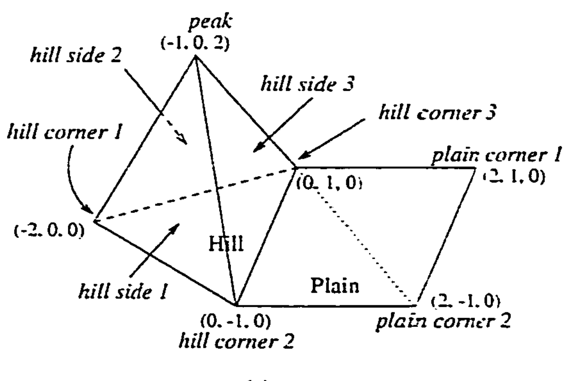
where S_t is a surface patch[1] in $\mathcal{T}$, which contains k triangles (denoted as t_{t_1}, t_{t_2}, ..., t_{t_k}), $h_{hyper} \in H_{hyper}$ is a hyperedge that has nodes h_{t_1}, h_{t_2}, ..., h_{t_k}, each of which is an element in H_t, such that by Equation 3.3, $t_{t_i} \overset{F_{g-obj}}{\longrightarrow} h_{t_i}$. Both S_t and h_{hyper} have attribute value a_h, which is the average normal vector of all triangles in S_t. To form a meaningful geometrical surface patch from a collection of triangles, H_{hyper} should be *connected.*

Figure 3.13 is an example of geometrical modeling by DAH. In Figure 3.13(a), a simple natural terrain is approximated by a triangular mesh: a hill is represented by a tetrahedron; a small piece of plain field at the side of the hill is represented by two triangles. Figure 3.13(b) shows the corresponding attributed hypergraph, and Figure 3.13(c) shows the net-like DAH. Table 3.1 gives the mappings of the elementary nodes and the hyperedge nodes from the geometrical descriptions (Figure 3.13 (a)) to the corresponding entities in the DAH (Figure 3.13 (c)) for the natural terrain. The attribute values for the nodes (for elementary nodes, they are the vertex coordinates, and for hyperedges, that are the normal vectors) are also shown in the table. Due to space limitations, the mappings of the triangle nodes and the full attribute lists for vertices and hyperedges are not shown.

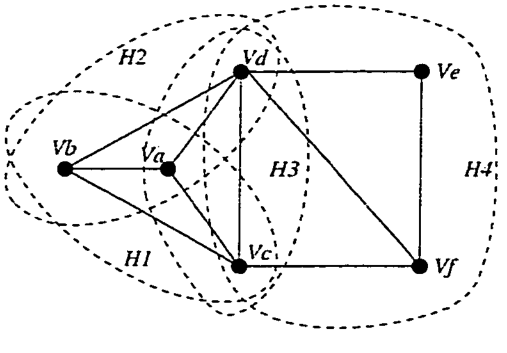
Modeling of Kinematic Transformations

Kinematic transformations include rotation, translation, scaling and their combi-

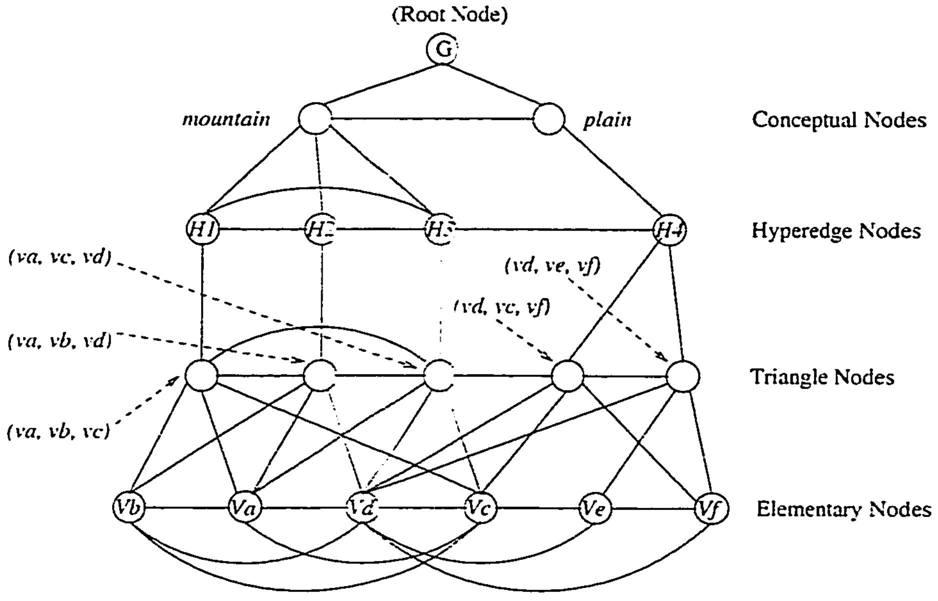
[1]By definition, a patch in the mesh is a collection of connecting triangles among which the differences of orientation are less than a pre-assigned threshold. Two patches are considered to be connected if they have at least one common edge.



(a)



(b)



(c)

Figure 3.13: An example of the DAH representation: (a) a natural terrain with a hill and a piece of plain land: (b) the attributed hypergraph representation; (c) the DAH structure.

Table 3.1: The geometrical features and the corresponding entities in the attributed hypergraph representation for a natural terrain.

Geometrical Feature	AH Entity	Attribute
peak	v_a	$(-1, 0, 2)^*$
hill corner 1	v_b	$(-2, 0, 0)^*$
hill corner 2	v_c	$(0, -1, 0)^*$
hill corner 3	v_d	$(0, 1, 0)^*$
plain corner 1	v_e	$(2, 1, 0)^*$
plain corner 2	v_f	$(2, -1, 0)^*$
hill side 1	H_1	$(-\frac{2}{\sqrt{21}}, \frac{4}{\sqrt{21}}, \frac{1}{\sqrt{21}})^{**}$
hill side 2	H_2	$(-\frac{2}{\sqrt{21}}, -\frac{4}{\sqrt{21}}, \frac{1}{\sqrt{21}})^{**}$
hill side 3	H_3	$(\frac{2}{\sqrt{5}}, 0, \frac{1}{\sqrt{5}})^{**}$
plain	H_4	$(0, 0, 1)^{**}$

Note: (*) values shown are the coordinates; (**) values shown are the orientations.

nations. If before and after the transformations the point sets are expressed in a fixed homogeneous coordinate system, all the transformations can be considered as matrix multiplications. Following the convention in CAD modeling, we represent a point in 3D space by $(x, y, z, 1)^T$ where x, y, z denote the three coordinates. Kinematic transformations can be written in the form of matrix multiplications. Translations, rotations and scalings are achieved by choosing different parameter sets in the 4×4 matrices.

The scaling and translation joint matrix T_S is in the form of:

$$T_S = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & w \end{pmatrix} \tag{3.5}$$

where d_x, d_y and d_z denote the translations in the directions of x, y and z respectively, and w is the scaling factor. The rotation matrix has the form:

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.6}$$

Since kinematic transformations are about the location changes of an object, they do not introduce structural change in AHR when information is abstracted from the realization level to the representation level. At the realization level, the transformations apply to the geometrical entities such as lines, corners, triangles or patches. In the AHR level, they apply to the geometrical attributes of vertices, edges or hyperedges which represent the shape. Such transformations are efficient to represent rigid motions.

With the definition of F_{g-mor} in $F_g = (F_{g-obj}, F_{g-mor})$, which maps the matrices T_S and R to the morphisms in the Ahp, we have:

1. scaling and translation mapping:

$$T_s \xrightarrow{F_{g-mor}} f_a \tag{3.7}$$

where T_s is a scaling and translation matrix given in Equation 3.5 and $f_a((x, y, z)) = (x + f_{a_x}, y + f_{a_y}, z + f_{a_z})$ is an attribute transition operator that operates on the geometrical attributes; when T_s is applied on a point (x, y, z) with parameter d_x, d_y, d_z and w as given in Equation 3.5, $f_{a_x} = \frac{(1-w)x+d_x}{w}$, $f_{a_y} = \frac{(1-w)y+d_y}{w}$ and $f_{a_z} = \frac{(1-w)z+d_z}{w}$.

2. rotation mapping:
$$R_s \overset{F_{g-mor}}{\longrightarrow} o_r \tag{3.8}$$
where R_s is a rotation matrix given in Equation 3.6 and $o_r((x, y, z)) = (x + f_{a_x}, y + f_{a_y}, z + f_{a_z})$ is an attribute transition operators that operates on the vertex or hyperedge attributes according to the rotation given by R_s. If R_s is applied on vertex (x, y, z) and has the matrix entries r_{ij} ($1 \leq i \leq 3$ and $1 \leq j \leq 3$) as shown in Equation 3.6, then $f_{a_x} = (r_{11} - 1)x + r_{12}y + r_{13}z$, $f_{a_y} = r_{21}x + (r_{22} - 1)y + r_{23}z$, and $f_{a_z} = r_{31}x + r_{32}y + (r_{33} - 1)z$.

In the previous definition of $F_g = (F_{g-obj}, F_{g-mor})$ that maps *Geo* to *Ahp*, it can be observed that each step to construct an attributed hypergraph with mapping F_{g-obj} or F_{g-mor} is a *one-to-one* mapping. Thus, they are *reversible*. By reversing those definitions, we can define the *inverse* mappings of F_{g-obj} and F_{g-mor}, denoted as F^{-1}_{g-obj} and F^{-1}_{g-mor}. The corresponding functor $F_g^{-1} = (F^{-1}_{g-obj}, F^{-1}_{g-mor})$ represent the process that back-projects an attributed hypergraph to a triangular mesh. Furthermore, $F_g \odot F_g^{-1} = I_{Ahp}$ and $F_g^{-1} \odot F_g = I_{Geo}$. Therefore, recalling Definition 3.6, *Geo* and *Ahp* are *equivalent*.

Theorem 3.3 Functor $F_g = (F_{g-obj}, F_{g-mor})$ preserves isomorphism.

[Proof]

Suppose that the RAG of the triangular mesh $\mathcal{T}$ is $G_t = (V_t, E_t)$, the AHR in the form of DAH is $G_a = (v_g, H_{base}, H_t, H_{hyper})$, where the *elementary graph* is the graph formed by $G_e = (v_g, H_{base})$. Functor F_g is the mapping from $\mathcal{T}$ to AHR: $Geo \xrightarrow{F_g} Ahp$.

From Equations 3.1 and 3.2, it is clear that $G_e \sim G_t$ since G_e simply duplicates G_t. For any morphism u defined on G_t, if $G_t \xrightarrow{u} G_t'$, by the definition $F_g =$

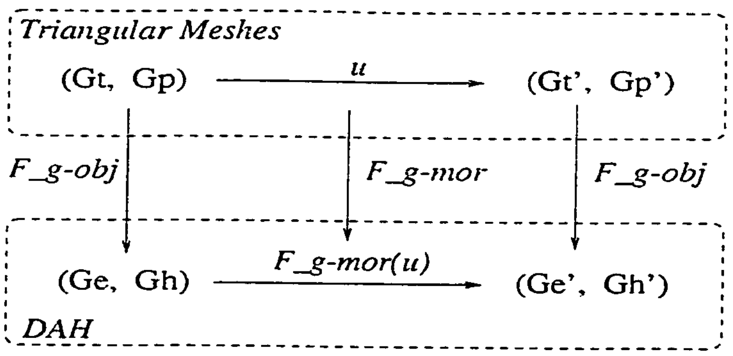


Figure 3.14: The relations between triangular mesh and attributed hypergraph with the kinematic functors.

(F_{g-obj}, F_{g-mor}), we have $G_a \overset{F_{g-mor}(u)}{\longrightarrow} G_a'$. If the *elementary graph* of G_a' is denoted as G_e', then $G_e' \sim G_t'$. When u is an isomorphism, $G_t \sim G_t' \sim G_e \sim G_e'$.

If on the layers above the RAG G_t, the base triangles and groups of triangles (called the patches) are written in the form of an attributed hypergraph $G_p = (V_T, E_P)$, and in the DAH representation the attributed hyperedges built upon G_e is expressed by $G_h = (H_t, H_{hyper})$, then $G_p \sim G_h$. Therefore, if $G_p \overset{u}{\longrightarrow} G_p'$, we have the corresponding $G_h \overset{F_{g-mor}(u)}{\longrightarrow} G_h'$. If $G_p \sim G_p'$, then $G_h \sim G_h'$.

As a result, if u is an isomorphism, $F_{g-mor}(u)$ is also an isomorphism: F_g preserves isomorphism.

□

Figure 3.14 illustrates the relations of the above graphs and mappings.

Modeling of Physical Transformations

Physical feature based representations describe deformable objects by physical parameters and state transitions. They use physical laws and constraints to govern the motions. For a physical process simulation, one only needs to consider: (1) the physical properties of the objects; (2) the physical laws applied to the objects; (3) the initial states and (4) the external forces applied. Then the new states of the objects at each time frame can be computed.

Similar to the kinematic transformations, the simulation of physical transitions calculates the shape and location changes of an object. The physical parameters do not change their forms across different representation levels during the abstractions of information. Physical feature descriptions are especially efficient to represent deformations of non-rigid 3D patterns.

In general, the object's physical state (in the form of physical parameters) at time t can be written as $S(t)$, and the state transition function is f. Given the initial state $S(0)$, function f calculates $S(T)$ where $T > 0$. For elastic objects, the form of f is normally one or a set of partial differetial equations [28, 86, 87].

Suppose that in the category *Phy*, each object is the vertex configurations of the physical parameters $C(t) = \{S_i(t) | 1 \leq i \leq n\}$ for a 3D pattern represented by a triangular mesh with n vertices. The morphisms in *Phy* are defined as triplets $u(t_1, t_2) = (C(t_1), f(t_1, t_2), C(t_2))$, which means that, with initial condition as $C(t_1)$ at t_1, by function $f(t_1, t_2)$, we can evaluate $C(t_2)$ at t_2 ($t_2 > t_1$) (i.e., $C(t_1) \xrightarrow{u(t_1,t_2)} C(t_2)$). Now we prove the following theorem:

Theorem 3.4 Triangular mesh based physical descriptions of 3D patterns together with the associated morphisms form a category *Phy*.

[Proof]
$u(t_1, t_2)$ is admissible since it is defined as a triplet $(C(t_1), f(t_1, t_2), C(t_2))$. Given two physical states $C(t_1)$ and $C(t_2)$, $u(t_1, t_2)$ is unique. For two morphisms $u(t_1, t_2) = (C(t_1), f(t_1, t_2), C(t_2))$ and $u(t_3, t_4) = (C(t_3), f(t_3, t_4), C(t_4))$, $u(t_1, t_2) = u(t_3, t_4)$ iff $C(t_1) = C(t_3)$, $C(t_2) = (t_4)$ and $f(t_1, t_2) = f(t_3, t_4)$.

For each object $C(t)$ in *Phy*, there is a unique identity morphism $I_{C(t)} = (C(t), f_I, C(t))$ where f_I is the linear identity function $f_I(C(t)) = C(t)$.

Given $t_1 < t_2 < t_3$, the composition of morphisms of $u(t_1, t_2)$ and $u(t_2, t_3)$ is: $u(t_2, t_3) \odot u(t_1, t_2) = (C(t_2), f(t_2, t_3), C(t_3)) \odot (C(t_1), f(t_1, t_2), C(t_2)) = (C(t_1), f(t_2, t_3) \odot f(t_1, t_2), C(t_3))$. Since the evaluations of the partial differential equations $f(t_1, t_2)$ and $f(t_2, t_3)$ are defined over a continuous domain, it is obvious that $f(t_2, t_3)$ can assume input as the output of $f(t_1, t_2)$. Therefore, $C(t_1) \xrightarrow{u(t_2,t_3)\odot u(t_1,t_2)} C(t_3)$.

From the above construction of compositions, it is immediate that: for $C(t_1)$, $C(t_2)$, $C(t_3)$ and $C(t_4)$, if $C(t_1) \xrightarrow{u(t_1,t_2)} C(t_2)$, $C(t_2) \xrightarrow{u(t_2,t_3)} C(t_3)$ and $C(t_3) \xrightarrow{u(t_3,t_4)} C(t_4)$, compound operators $u(t_2, t_3) \odot u(t_1, t_2)$ and $u(t_3, t_4) \odot u(t_2, t_3)$ can be constructed, such that: $C(t_1) \xrightarrow{u(t_2,t_3)\odot u(t_1,t_2)} C(t_3) \xrightarrow{u(t_3,t_4)} C(t_4)$ and $C(t_1) \xrightarrow{u(t_1,t_2)} C(t_2) \xrightarrow{u(t_3,t_4)\odot u(t_2,t_3)} C(t_4)$.

It is also obvious that the identity morphism $f_I(C) = C$ always compose with $u(t_1, t_2)$ to give $f_I \odot u(t_1, t_2) = u(t_1, t_2)$ and $u(t_1, t_2) \odot f_I = u(t_1, t_2)$.

By showing that *Phy* has all the properties of a category, we demonstrate that the physical descriptions of 3D patterns together with the morphisms derived from the partial differential equations form a category.
□

The DAH $H = (X, H_{base}, H_t, H_p)$ for the physical description is formed by the definition of the functor $F_p = (F_{p-obj}, F_{p-mor})$ which maps the physical configurations from triangular mesh to the AH category. Suppose that the triangular mesh for finite element analysis is in the form of its RAG $G = (V_t, E_t)$ where the vertex set V_t corresponds to the vertex set in the mesh, and edge set E_t corresponds to the edge set in the mesh. The mapping functor F_p is defined as:

1. elementary node mapping:
$$v_t \overset{F_{p-obj}}{\longrightarrow} v_p \tag{3.9}$$
where $v_t \in V_t$ and $v_p \in X$, both with attribute value $a_v = (p_{v_t}, F_{v_t}, M, \gamma, \mu_{v_t}, \delta\epsilon_{p_{v_t}})$ denoting the physical status associated with vertex v_t.

2. elementary edge mapping:
$$e_t = \{v_{t_1}, v_{t_2}\} \overset{F_{p-obj}}{\longrightarrow} h_{base} = \{v_{g_1}, v_{g_2}\} \tag{3.10}$$
where $e_t \in E_t$, $h_{base} \in H_{base}$ such that by Equation 3.9, $v_{t_i} \overset{F_{p-obj}}{\longrightarrow} v_{g_i}$ $(i = 1, 2)$; e_t and h_{base} both have attribute value a_e denoting the directional vector for the edge e_t.

3. triangle mapping:
$$t_t = \{v_{t_1}, v_{t_2}, v_{t_3}\} \overset{F_{p-obj}}{\longrightarrow} h_t = \{v_{g_1}, v_{g_2}, v_{g_3}\} \tag{3.11}$$
where t_t is a triangle in the mesh with vertices v_{t_1}, v_{t_2} and v_{t_3}, $h_t \in H_t$ is a hyperedge contains vertices v_{g_1}, v_{g_2} and v_{g_3}, such that by Equation 3.9, $v_{t_i} \overset{F_{p-obj}}{\longrightarrow} v_{g_i}$ $(1 \le i \le 3)$. t_t and h_t both have attribute value $a_t = \vec{t_t}$ denoting the normal vector of the triangle t_t.

4. hyperedge mapping:
$$S_t = \{t_{t_i} | 1 \le i \le k\} \overset{F_{p-obj}}{\longrightarrow} h_p = \{h_{t_i} | 1 \le i \le k\} \tag{3.12}$$
where S_t is a surface patch consisting of k triangles (denoted as t_{t_1}, t_{t_2}, ..., t_{t_k}) in the mesh. $h_p \in H_p$ is a hyperedge that has nodes h_{t_1}, h_{t_2}, ..., h_{t_k}, each of which is an element in H_t, such that by Equation 3.11, $t_{t_i} \overset{F_{p-obj}}{\longrightarrow} h_{t_i}$. S_t

and h_p both have attribute value a_h denoting the average normal vector of all triangles in S_t. h_p itself consists of a connected attributed graph.

5. operator mapping:

$$(V_t(t_0), D(t_0, t_1, t_2, ..., t), V_t(t)) \xrightarrow{F_{p-mor}} (X(t_0), (o(t_0), o(t_1), o(t_2), ..., o(t)), X(t)) \tag{3.13}$$

where $D(t_0, t_1, t_2, ..., t)$ is the state transition function f with initial state at t_0 and evaluated at $t_1, t_2, ..., t$, $V_t(t_0)$ and $V_t(t)$ are the states of the vertex set V_t at time t_0 and t respectively, $X(t_0)$ and $X(t)$ are the attributed vertex sets of the DAH at time t_0 and t respectively. $((o(t_0), o(t_1), o(t_2), ..., o(t))$ is the set of AH operators that are applied to $X(t_0)$ and transform it to $X(t)$ $(t_0 < t_1 < t_2 < ... < t)$.

If the AH category *Ahp* is formed by the category of physical description *Phy* by $Phy \xrightarrow{F_p} Ahp$, then similar to F_g, the process of F_p is also reversible. Therefore, *Phy* and *Ahp* are *equivalent.*

By Equation 3.13, it can be seen that in the scope of modeling physical transformations, the physical transitions given by $D(t_0, t_1, ..., t)$ will not introduce structural changes to the RAG of the triangular mesh. Similarly, the corresponding morphisms $((o(t_0), o(t_1), ..., o(t))$ are all attribute transitions. In other words, if physical transition u maps RAG G_1 to G_2 in Phy, and the corresponding representation in Ahp is $F_{p-obj}(G_1) \xrightarrow{F_{p-mor}(u)} F_{p-obj}(G_2)$, then $G_1 \sim G_2 \sim F_{p-obj}(G_1) \sim F_{p-obj}(G_2)$. Therefore, $F_p = (F_{p-obj}, F_{p-mor})$ preserves isomorphism.

Modeling of Behavioral Factors

The representation on the behavioral factors of the object has not been taken into consideration by researchers until the recent years, when animations and simulations that are less dependent on user inputs become increasingly important. As argued in Chapter 2, the overlook of the behavioral factor in modeling is by large due to the lack of representation and computation power of symbolic information in the early systems.

The behavioral features embedded in an object representation in which we are interested include:

- knowledge about ego-motion of the object, such as fixed motion patterns;
- internal constraints on motion, such as fixed posture or orientation (they maybe posed in accordance with other kinematic or physical constraints);
- self-intelligence of the object (in such case it is usually called an "intelligent agent") in the form of sets of rules that govern the motions.

These features are difficult to represent with traditional geometry or physics oriented modeling frameworks. However, in an attributed hypergraph, the intrinsic data types are entities and their relations. The object's geometrical, kinematic and physical features are characterized by entities with attributes in an attributed hypergraph. Therefore, the "knowledge", "constraints" or "intelligence" of this object can be interpreted by high-order relations among these entities. In a graph language, these relations can be expressed by:

- attributed vertices, edges and hyperedges that characterize geometrical, kinematic and physical features;

- associations of the above entities in the form of high level hyperedges;
- operators applied to the above attributed graph entities; and
- rules defined on the compositions of the operators.

The corresponding mapping functor F_b which forms the behavioral factor representation from the above features in AH category is:

1. elementary node mapping as defined in Equation 3.1 or Equation 3.9;
2. elementary edge mapping as defined in Equation 3.2 or Equation 3.10;
3. triangle mapping as defined in Equation 3.3 and Equation 3.11;
4. geometrical transformation mapping as defined in Equations 3.7 and 3.8;
5. operator mapping as defined in Equation 3.13;
6. process abstraction from the item #4 and item #5 to a definite operator sequence $\{o_1, o_2, ..., o_n\}$ where $o_i (1 \leq i \leq n)$ is either a geometrical transformation on AH (f_a or o_r) or a primary operator on AH (o_p).

Chapter 4

AHR Applications in Augmented Reality

4.1 Introduction

The attributed hypergraph representation (AHR) presented in the previous chapter provides an efficient and practical methodology for vision based object modeling, which has broad applications in computer vision and graphics. In this chapter, the application of AHR in augmented reality (AR) is demonstrated. AR is an excellent example for showing the flexibility of AHR based modeling. It first employs computer vision technologies to estimate the 3D geometry of real scenes, and then facilitates computer graphics technologies to augment the synthesized models and to present them in a way similar to that of virtual reality.

In most vision based 3D scene analysis methodologies, image processing algorithms are first applied to extract imaging features from the digitized images to reduce the amount of data to be processed. Normally these features include corners, lines, contours, faces or regions, which are closely related to the actual shapes, presenting partial geometrical properties of the scene. Furthermore, if multiple views of a fixed scene are available, range features (which are considered to be $2\frac{1}{2}$D) can be recovered by vertex triangulation. Since these features are associated with each other via geometrical or topological relations, they can be organized into an attributed graph or hypergraph. Therefore, it is possible for us to build a partial AHR induced by the features of a 3D scene, and to incrementally synthesize a full AHR through the data from inexpensive sensors such as CCD cameras. On the graphics side, if a 3D scene has been modeled by an AHR, it would be easy to combine it with the models of virtual objects. We can change the shape of any object from a real scene or from a virtual source, or change the appearance (such as textures, shadows or reflections) of the combined scene.

A typical multiple-view vision system used for AR construction is shown in Figure 4.1 in which t_0, t_1 and t_2 specify three different camera poses (though, in most systems, the number of views may vary). c_0, c_1 and c_2 are the three image coordinate systems associated with the three camera poses respectively; c denotes a fixed common world coordinate system; and m_0, m_1 and m_2 denote the image points for the object point X_g on the three image planes respectively. The task of the vision system is to obtain the 3D geometry of the scene in c from the images acquired at t_0, t_1 and t_2.

In the following sections, the methodology of vision based AR is presented. Several key issues in computer vision, namely, feature detection, camera calibration, 3D geometry recovery, and incremental AHR synthesis are addressed, followed by the algorithm of AHR augmentations. Finally, experimental results are provided.

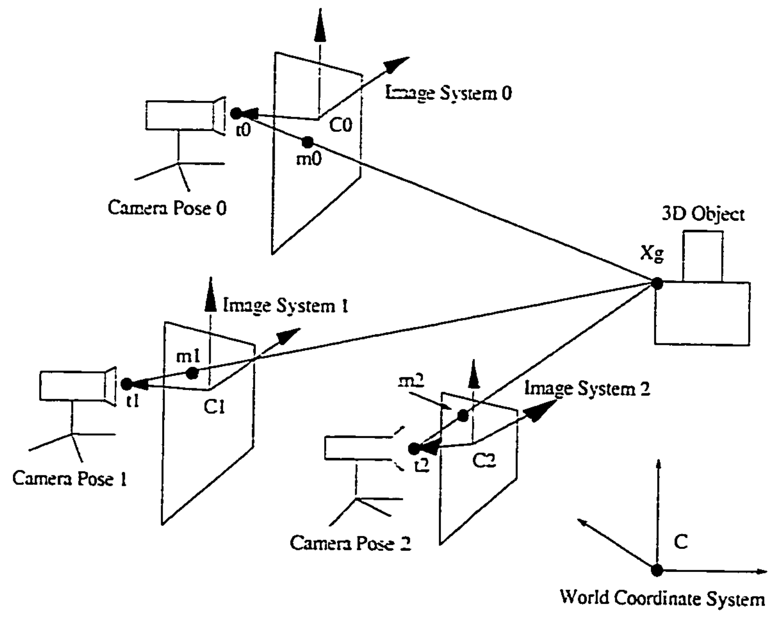


Figure 4.1: The multiple-view system for 3D model synthesis.

4.2 2D Feature Extraction

All camera based image processing methodologies begin with 2D images, each of which typically contains several hundred thousands of pixels. It is neither practical nor productive to carry on image understanding directly on the pixel level. Thus, certain "features" should be extracted in order to obtain higher level information.

To analyze the shape of simple objects presented in 2D images for shape reconstruction, edge based features such as lines, corners and curve segments are important. The feature extraction procedure applied in this thesis research consists of four parts: 1) edge detection; 2) edge segmentation; 3) edge classification and 4) feature grouping.

4.2.1 Edge Detection

Effective edge detection is required by most image understanding applications. Traditional methods such as Hough transform, heuristic search-inference and many others have a common shortcoming: they are all computationally intensive largely because they mainly rely on precise pixel by pixel calculations. The edge detection method adopted here is from [41], which is based on the perceptual organization [62] of descriptive edge features. The edge tracking procedure to find the edge pixels is Zuker and Prager's relaxation based edge-linking method [73, 108]. It tracks edge segments then joins and fits them into appropriate forms of line or curve structures according to their topological and geometrical properties.

The raster scan process brings about a disadvantage of the algorithm. Despite its fast speed, it may result in different traces merging together. This problem is solved by further segmentations of the edge detection result with additional geometrical criteria.

4.2.2 Edge Segmentation

The above edge detection algorithm provides us with edge traces on which each point is described by four attributes:

- (X, Y): the 2D coordinates of the point in the image;
- *transient*: a Boolean value indicating discontinuity of the trace at the point;
- *slope*: the slope obtained from a five-point average;
- *curvature*: the rate change of the slope.

If a point is on a smoothly connected edge trace, its *transient* should stay zero; otherwise, it is considered as a special point along the edge trace. All corners, crossing points, edge breaking points and local spurious noises most likely have non-zero transient values. Edge traces are separated into different segments at these points. Segments with lengths smaller than a given threshold are considered as noise and removed.

4.2.3 Edge Classification

There are two types of edge traces that we are interested in:

- *straight lines*

  They are the key features to characterize the shapes, especially polyhedral objects. Straight lines and their junctions are the major sources to form a triangular mesh. Furthermore, it is normally easy to locate the junctions of straight lines accurately in 2D images, thus they are ideal for image based camera calibration.

- *elliptical curves*

  In the pin-hole camera model, the perspective projection of a 3D circle is a full or partial ellipse. Though we use triangular mesh for general shape approximation on the range data, well-detected ellipses in 2D images will greatly enhance the accuracy of the mesh approximation on 3D circles. For camera calibration, they can be used to verify the estimated pose, which is more reliable than verification through the junction points.

The extracted edge segments are classified into the above two categories according to the changes in the slope. In a real-world image, however, edge traces are

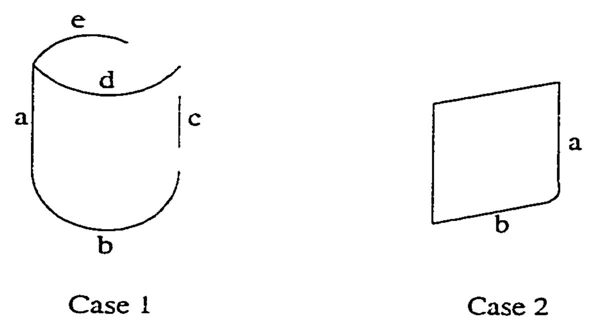


Figure 4.2: Two kinds of smooth connections: 1) an ellipse fraction and a virtual edge; 2) two straight lines in one pseudo-segment.

affected by the digitization error. noise and parameters. Sometimes the slope alone is not sufficient to distinguish a line segment from an elliptical one. For example, a line and a curve are connected so smoothly that the change of transient may not be detected. We call such a segment a *pseudo-segment*. Figure 4.2 gives two examples. Here, case 1 is a smooth connection of an ellipse fraction b and a virtual edge a. Case 2 is a smooth connection of two straight lines (pseudo-segment $\widehat{ab}$).

In both situations, the slope associated with such curves and lines does not help much to partition the *pseudo-segment* into two. To find the connection point, the curvature has to be used to assign segments into a line class or a curve class. The curvature of a straight line should be zero while the curvature of a segment of elliptical curve should be non-zero and only change slowly.

4.2.4 Feature Grouping

The objective of feature grouping is to clutter 2D features in an image into one or more groups according to their structural or topological relations, each of which could be associated to the 3D object to be analyzed. Feature grouping effectively reduces the search space for feature matching used in camera calibration and model synthesis. It can be formalized as follows:

Definition 4.1 Given a set of 2D features F, the grouping procedure C outputs a set of possible connected groups $\{F_i\}$ each of which can be related to one or more possible objects $\{obj_j\}$, i.e.,

$$\begin{array}{ll} C: & F \rightarrow \{F_1, \cdots, F_n\} \\ s.t. & \bigcup_{i=1}^{n} F_i = F \quad and \quad F_i \Rightarrow \{obj_j\} \end{array} \tag{4.1}$$

Here we restrict the elements of F to one of the two features: a) fitted straight lines or b) elliptical curves.

Line Fitting and Corner Grouping

Corners are the junctions of straight lines or elliptical curves. To accurately locate a corner, the straight line segments are first fitted and the equation of each line is calculated in the form of $x/a + y/b = 1$.

From a line segment, we can estimate its parameters accordingly. If its $X-intercept$ a and $Y-intercept$ b are not equal to zero, we use the *median of intercepts* method [54] to fit the line. Suppose that there are N points in a line fitting set, then any two points i and j can form a line and render an (a_{ij}, b_{ij}) pair. There are at most $N(N-1)/2$ pairs each of which can be described as

$$a_{ij} = \frac{x_j y_i - x_i y_j}{y_i - y_j} \quad (y_i \neq y_j), \qquad b_{ij} = \frac{x_i y_j - x_j y_i}{x_i - x_j} \quad (x_i \neq x_j), \tag{4.2}$$

where $1 \leq i \leq j \leq N$. Then a and b can be obtained by:

$$a = median\{a_{ij}\}, \qquad b = median\{b_{ij}\}.$$

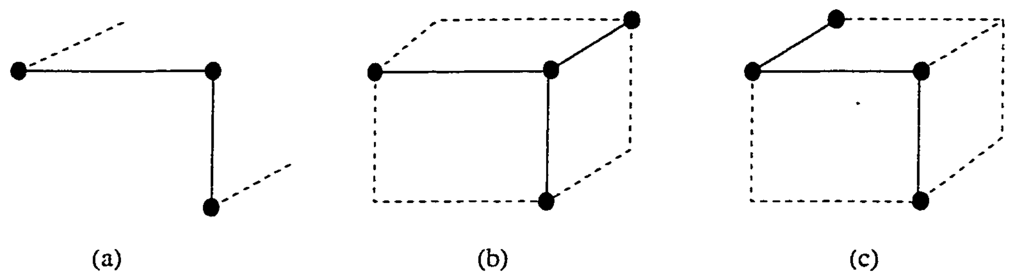


Figure 4.3: Types of feature groupings: (a) a 3-point L-shape feature; (b) a Y-shape 4-point feature grouping; (c) a concatenated 4-point feature grouping.

The following rules are posed to obtain corners from intersecting lines:

- a corner must be located on the *extension* or at either end of each line in the explored set to avoid having $T-junctions$;
- two parallel or nearly parallel lines in a set do not form a corner;
- if several intersections are found to be very close to each other, their average position is taken as the corner position.

Once the corners are obtained, it is easy to find groups of 3-point and 4-point features. A typical 3-point feature is one with L (or V) shape (Figure 4.3 (a)). A 4-point feature is formed with two 3-point features linked by a common line segment. There are two types of 4-point feature:

- a Y-shape feature where three junction points are connected to the same junction point (Figure 4.3 (b));
- a concatenated feature where four junction points are connected by edges to form a chain (Figure 4.3 (c)).

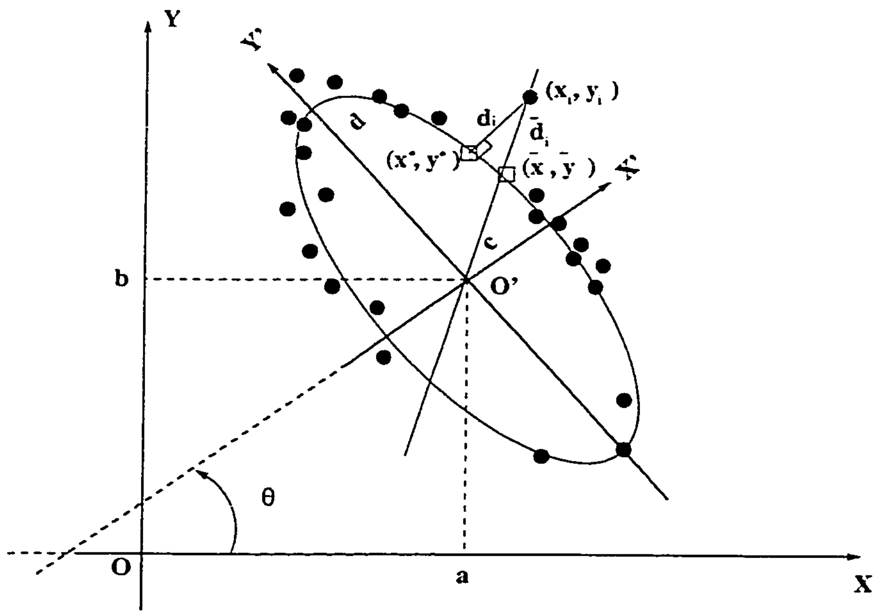


Figure 4.4: Description of ellipse fitting and ellipse parameters.

Elliptical Curve Grouping and Ellipse Fitting

The equation applied for ellipse fitting is:

$$\frac{[(x-a)\cos\theta + (y-b)\sin\theta]^2}{c^2} + \frac{[-(x-a)\sin\theta + (y-b)\cos\theta]^2}{d^2} = 1 \qquad (4.3)$$

where (a, b) is the center of the ellipse, c, d ($c > 0$, $d > 0$) are its axes, and θ is its rotation angle (orientation) — the angle between the x axis and the short axis of the ellipse. When $c = d$, the ellipse becomes a circle. Figure 4.4 illustrates these parameters.

Accurate estimations of the five parameters of an elliptical curve (the center coordinates, the major and minor axes, and the orientation) are required in various machine vision related problems. Many techniques have been developed to deal with elliptical parameter estimation. However, most of them are based on the local fitting concept, i.e. they try to estimate the five parameters provided with a segment of an ellipse. Due to the presence of short curve segments and local spurious noise, the fitting often yields a hyperbola or other curve forms. Different from those methods, the concept of global fitting introduced in [105] is applied: before fitting a selected curve segment, the entire curve set is searched to see if it can be grouped with another one to form an elliptical curve grouping in compliance with their 2-D geometrical and topological constraints. This procedure effectively reduces the influence of noise and the possibility of fitting points from different curves.

For elliptical curve grouping, we use the following attributes obtained from the analysis of curve segments:

- identity index I of a curve segment;
- curve length L;
- section number S to which a curve segment belongs (4 sections corresponding to 4 quadrants in the $X - Y$ coordinate system);
- direction D of a curve segment;
- curve starting and ending points (X_s, Y_s) and (X_e, Y_e).

The fitting procedure begins with selecting a curve segment from the curve set, and then searching the starting and the ending points of its neighboring segments. Two curve segments are grouped together if they have their end points close enough, and if the following conditions of curve adjacency are satisfied:

1. two curve segments which are in the same section can be grouped if and only if their terminating points intersect;

Table 4.1: Grouping restrictions of two curve segments S_1 and S_2.

S_1 Section	S_2 Section	Restrictions
I	II	$S_2.X_{max} < S_1.X_{min}$
I	III	$S_2.X_{max} < S_1.X_{max} \wedge S_2.Y_{max} < S_1.Y_{max}$
I	IV	$S_2.Y_{max} < S_1.Y_{min}$
II	III	$S_2.Y_{max} < S_1.Y_{min}$
II	IV	$S_1.X_{max} < S_2.X_{max} \wedge S_2.Y_{max} < S_1.Y_{max}$
III	IV	$S_1.X_{max} < S_2.X_{min}$

2. two curve segments which are in different sections can be grouped if and only if they satisfy the conditions described in Table 4.1.

A popular fitting method is by minimizing the *linear least-squares* (LLS) error. It is simple and usually gives good results. However, its computational cost is high. Since the possibility of getting a hyperbola solution is greatly reduced after the curve grouping, a fast iterative algorithm [105] is adopted. It starts with an initial guess for the parameters, then fits the equation to the data recursively until the error requirement is satisfied. The *Newton* iteration method is applied.

4.3 Camera Calibration

To correctly reconstruct a 3D scene using 2D images taken from CCD cameras, we need to know the relative positions of the different cameras. This requires us to compute the *camera pose* for each of the images. It is essential for 3D model reconstruction that the camera poses be precise. Many technologies have been developed to locate and/or track the camera positions, with the aids of various data sources. Existing commercial systems use external devices, such as magnetic field based sensors, sonar sensors or laser range finders to measure the relative position between the landmark and the camera. Nevertheless, as observed in Chapter 2, calibration systems based on the external sensors correspond to open-loop controllers in which the calibration data and the image data are isolated. Their high costs also make them impractical in many expense-sensitive applications. Many researchers in computer vision areas have attempted to calibrate the camera pose directly using the data from the 2D images [50]. Though much effort has been made, the problems of accuracy, speed and robustness have not been completely solved.

In this thesis, a camera calibration algorithm directly based on the feature groupings in 2D images is implemented. The principle of the method is to use a simple-shaped object with known 3D measurement that is visible in the scene as the visual landmark to compute the camera pose. The correspondences between the 2D feature groupings and known 3D model feature of the landmark are used to compute the camera pose based on photographic equations. An intuitive advantage of the method is that the "open-loop" mentioned above can be closed by back-projecting the camera pose to the image for automatic confirmation.

4.3.1 Hypothesis Directed Matching

Feature matching in camera calibration attempts to establish correspondences between: (1) feature groupings extracted from 2D images, and (2) the 3D model features in the landmark model base. In general, feature matching requires exhaustive search and is recognized as an NP-complete problem. To speed up the matching, we use a hypothesis directed search based on a modified version of the constellation matching algorithm [98]. Hypothesis generated from the observed feature groupings is used as heuristics. For each matching hypothesis, one or more corresponding hypothetical poses are calculated with the steps given in Section 4.3.2. The pose results are verified by back-projecting the 3D features of the landmark to the 2D image. Then the initial hypothesis is evaluated with the following criteria:

- *plausibility*:
  a threshold on the total number of matched features to determine the acceptability of the hypothetical matching;
- *reliability*:
  a measure based on the total back-projection error with a weighting score from the feature grouping itself.

In our implementation of the hypothesis directed matching algorithm, the plausibility threshold is usually 4 to 8 pixels for general polyhedral landmarks, and the reliability threshold on matching error is 2 pixels per matched image feature.

4.3.2 Camera Pose Determination

Pose determination algorithms employing analytic or numerical solutions can be found in [26, 37]. Among the popular ones are the 3-point, 4-point and 6-point approaches. We adopt the approach suggested in [37], in which: (1) the pin-hole camera model is assumed; and (2) the solution of a quartic polynomial is required, thus we will obtain up to 4 candidate solutions.

It is observed through experiments that approaches with multiple solutions are more preferable than those that generate a unique solution: in many cases, multiple solution methods can yield more than one *correct* solutions but with slight differences [72]. Then we can apply one or a combination of the following techniques to make use of the redundant information:

- prune the solutions and select only the correct one(s) or the best one from a set of candidate solutions obtained from one combination of the matched features;
- use different combinations of the matched features to obtain different but all *correct* poses;
- compute the translational inconsistency measure and orientational inconsistency measure to evaluate the quality of the matched feature set;
- use the pose averaging algorithm described in [72] to improve the accuracy of the result.

A pose verification and refinement procedure is employed to select and refine the *correct* solution(s) for each possible matched feature combination. Given one of the possible pose solutions in the form of a rotation matrix R and a translation vector T, it is straightforward to obtain the inverse transformation which is then used to project the entire object model from the 3D world coordinate system to the image. In the image coordinate system, the back-projection error is calculated based on the discrepancy between the extracted image features and the corresponding projected 3D model features. The back-projection error, the number of matched feature groupings and their pre-calculated weighting scores are then combined into a plausibility measure. The candidate pose with the highest plausibility among all solutions in a matched feature combination is selected as the best correct solution for such combination.

Suppose that a set of n correct poses, $(R_i, T_i)|(1 \leq i \leq n)$, are obtained from all possible matched feature combinations. The translational and orientational inconsistency measures are defined as the following:

Definition 4.2 The translational inconsistency d of a set of poses (R_i, T_i) $(1 \leq i \leq n)$ is defined as the *relative average mutual distance of the translations*:

$$d = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} |T_i - T_j|}{\binom{n}{2}} \tag{4.4}$$

Definition 4.3 The orientational inconsistency number c_R of a set of poses (R_i, T_i) $(1 \leq i \leq n)$ is defined as the *average distance amongst the rotation matrices*:

$$c_R = \frac{1}{\binom{n}{2}} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\sum_{k=1}^{3} \sum_{l=1}^{3} (R_{i_{kl}} - R_{j_{kl}})^2}{9} \tag{4.5}$$

Thresholds for the d and c_R can be set depending on the applications to exclude cases that the landmark is at ill-conditioned positions.

Though the pose calculation is accurate over a wide range of viewing conditions, with the redundant information that results in more than one correct and consistent camera poses, the accuracy can be further improved by a pose averaging approach [72].

The average of the translations $\bar{T}$ is simply computed by:

$$\bar{T} = \frac{1}{n} \sum_{i=1}^{n} T_i \tag{4.6}$$

It should be noted that:

$$\bar{T} = \min_{T} \sum_{i=1}^{n} |T_i - T|^2 \tag{4.7}$$

The average of several rotation matrices should preserve the orthogonal property. Since the matrix stands for the orientations, a rotation matrix R_i can be decomposed into three Euler angles θ, ψ and ϕ:

$$R_i = \begin{pmatrix} c_2c_3 - c_1s_2s_3 & -c_2s_3 - c_1s_2c_3 & s_1s_2 \\ s_2c_3 + c_1c_2s_3 & -s_2s_3 + c_1c_2c_3 & -s_1c_2 \\ s_1s_3 & s_1c_3 & c_1 \end{pmatrix}, \tag{4.8}$$

and

$$c_1 = \cos\theta_i, \quad c_2 = \cos\psi_i, \quad c_3 = \cos\phi_i,$$

$$s_1 = \sin\theta_i, \quad s_2 = \sin\psi_i, \quad s_3 = \sin\phi_i$$

Then the problem of averaging matrices becomes the problem of averaging angles. Correct results cannot be obtained by simply taking the arithmetic average due to the periodic property of angles. To solve the problem, a vector $\vec{v}$ is constructed in 3D space for each three-angle set:

$$\vec{v_i} = a_ix + b_iy + c_iz$$

$$a_i = cos\theta_i, \quad b_i = cos\psi_i, \quad c_i = sin\phi_i$$

Then an average angle set is obtained from the average vector:

$$\bar{\vec{v_i}} = \bar{a}x + \bar{b}y + \bar{c}z$$

$$\bar{a} = \frac{1}{n}\sum_{i=1}^{n} a_i, \quad \bar{b} = \frac{1}{n}\sum_{i=1}^{n} b_i, \quad \bar{c} = \frac{1}{n}\sum_{i=1}^{n} c_i$$

$$\bar{\theta} = arccos\ \bar{a}, \quad \bar{\psi} = arccos\ \bar{b}, \quad \bar{\phi} = arcsin\ \bar{c}$$

Finally, an average rotation matrix can be constructed from $(\bar{\theta}, \bar{\psi}, \bar{\phi})$ with Equation 4.8.

4.4 3D Geometry Recovery

The task of 3D shape reconstruction in AR applications resembles 3D shape modeling in computer vision: given a series of calibrated 2D views, recover the 3D geometry of the scene. A considerable amount of work in computer vision has been devoted in this area [8, 50, 61, 75, 99]. Most of these approaches can be categorized as volume intersection technique for geometrical reconstruction, involving vertex or volume triangulation and then incremental synthesis. The method applied in this thesis is the stereo algorithm from [61], after which the models are converted into AHR [99] for incremental synthesis and AR integration.

4.4.1 Adaptive Stereo Matching of 2D Features

In multiple-view model synthesis, one of the most difficult tasks is the matching of features or feature groupings [65, 107] from different images. It is mostly infeasible to search the entire feature set in a pair of images for possible matchings. Here the matching of the feature groupings in two images is guided by the following constraints:

- the epipolar line geometry;
- the structural characteristics of the feature groupings;
- the probability of coincidence of the feature groupings obtained from one image in other images.

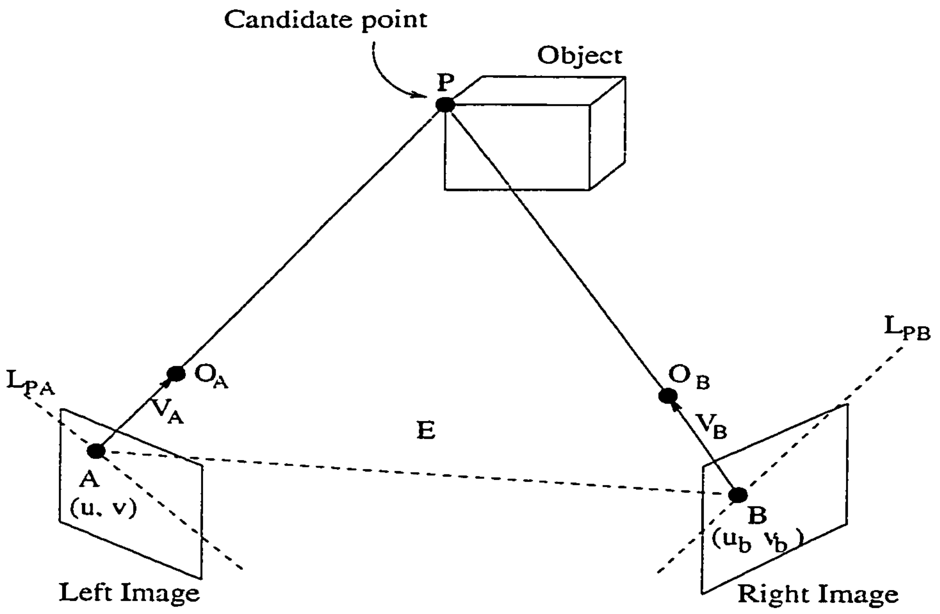


Figure 4.5: The epipolar line constraints in stereo matching.

Epipolar Line Constraints

An epipolar line is defined as the intersection of the epipolar plane with an image plane. To apply this constraint, the location of the image point corresponding to the object point must be known exactly in either one of the two images. Then it is used as the constraint to prune the search space of the corresponding point in the other image.

In Figure 4.5, let P be a point on the object, A be its image point on the left image plane, and B be the point on the right image plane. The focal points of the two cameras are O_A and O_B respectively. The epipolar lines L_{PA} and L_{PB} are the rays V_A and V_B projected on the left and right image planes respectively, where V_A and V_B are the directional vectors of the light rays passing through $\overline{(O_A, A)}$ and $\overline{(O_B, B)}$ respectively.

Point P, O_A and A are on the same line defined by vector V_A. Likewise, P,

O_B and B are on the same line defined by V_B. Thus points P, A, B, O_A and O_B are all on the same plane E, which is called the epipolar plane that intersects the left image plane at L_{PA} and the right image plane at L_{PB}. This is the geometrical principle for the epipolar line calculation.

Given a known image point $B = (u_b, v_b)$ on the right image plane, if the 3D coordinates of O_A, O_B and the two camera poses H_A, H_B are known with respect to a common reference, E can be computed from O_A, O_B and B. Then L_{AP} can be determined by the intersection of E and the left image plane. Therefore, a 3×3 matrix M from V_A, V_B, O_A and O_B is computed as:

$$M = \begin{pmatrix} V_A.x & V_B.x & O_A.x - O_B.x \\ V_A.y & V_B.y & O_A.y - O_B.y \\ V_A.z & V_B.z & O_A.z - O_B.z \end{pmatrix} \tag{4.9}$$

Let the determinant of M be zero to obtain the line equation of L_{AP}. If we express V_B by O_B and B, V_A by O_A and point (u, v), then L_{AP} has two variables u and v in the form of:

$$\alpha_{AP}u + \beta_{AP}v + \gamma_{AP} = 0 \tag{4.10}$$

where · α_{AP}, β_{AP}, γ_{AP} are determined from O_A, O_B, H_A, H_B and B;
· H_A, H_B are the homogeneous matrices that signify the two camera poses.

Thus, with the epipolar line constraint, given the coordinates of a point B on the right image, we can always find the correlated epipolar line on the left image. The matching problem is then simplified to the task of finding the matched feature along a line instead of from the entire image.

It should be noted that normally on or around a determined epipolar line, there are many feature points that can be considered as candidates if certain tolerances are included to accommodate the influence of noise and rounding errors. Hence, other proper matching criteria and/or constraints are important.

2D Feature Grouping Constraints

Since there could be hundreds of 2D feature points in an image, there could still be dozens of matching candidates subject to epipolar line constraints. Therefore, we need to evaluate the acceptability of a hypothetical matching of a pair of feature points from two images based on some invariant features associated with them. To address this problem, a cost function is introduced which returns a plausible measurement indicating the efficacy of a potential match [33]. A lower value would suggest a more acceptable pairing. If, however, the cost exceeds a specified threshold, the matching is deemed to be unacceptable and therefore rejected.

This rejection condition makes it possible to evaluate individual characteristics of a pair of feature points for their separate contributions to the overall cost penalty. After each attribute has been checked, the summed up cost can be tested against the pre-set threshold. The cost is associated with the attributes of the 2D features, which include:

- 2D coordinate offsets in the images;
- relative degrees of feature points (i.e., the number of attached lines to the corner feature);
- lengths of the attached lines;
- angles between the attached lines:
- connectivity of points with respect to already matched feature points.

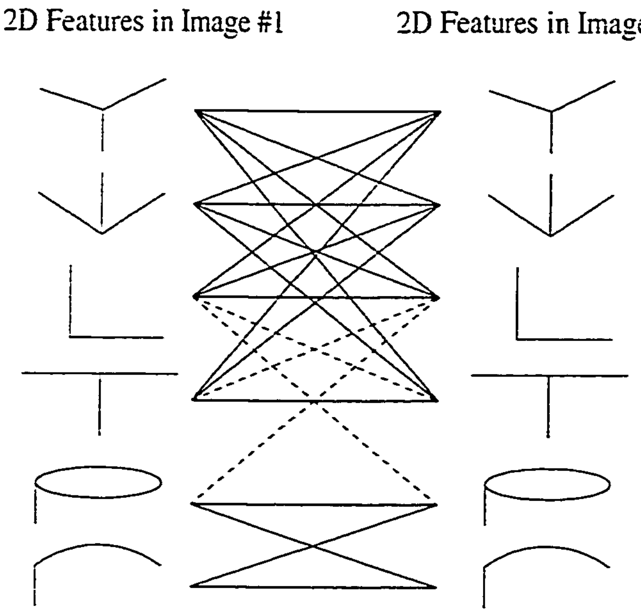


Figure 4.6: A bipartite graph showing the feature grouping constraints in stereo matching.

In most applications, between two consecutive images, there are only a limited degree of rotations and translations. In our AR experiments, the rotations vary from 10^o to 15^o. It is expected that the corresponding feature pairs on the two images mostly remain within the same local area. If the (x, y) offset of a feature is out of the local neighborhood, a penalty is imposed. If there are lines associated with the feature point being occluded or not identified properly due to noise, the difference between the number of associated lines of the same point in the image pair should not exceed one for acceptance. Otherwise, a penalty cost is assigned on the relative difference. Figure 4.6 shows the bipartite graph of the matching plausibilities between different 2D feature groupings. In the figure, solid lines stand for most plausible hypotheses while dashed lines indicate very little possibility for the matching pairs.

In addition, the lines connected to the points being compared should not have their angles and lengths changed dramatically between consecutive images. Finally, part of the total matching cost can be attributed to their relations with other feature points that are already matched. For example, when two feature points are compared, if they share a matched pair of points connected to them with the same connection type, their plausibility of matching is much higher. A bonus can be subtracted from the matching cost.

Back-projection Constraint

In AR applications, if the number of images of the scene is more than two, another constraint can be applied to calculate the plausibility of a matching hypothesis. Since we assume that between each two consecutive frames, the displacement of the camera is relatively small, it is reasonable to expect that if a pair of 2D feature points are matched in two images, their corresponding 3D point is likely to appear in the consecutive images.

The constraint can be verified by back-projection. First, we assume that the feature pair is matched. Then by vertex triangulation (refer to Section 4.4.2), the 3D coordinates of this point in the world system is computed. Since the camera poses of the image sequence are already obtained from the landmarks, the 3D coordinates could then be back-projected onto the images that immediately precede and succeed to the two images where the vertex triangulation has been performed. In the two extra images, 2D feature points are searched within a small area around the back-projected location. If no feature point is found around the area in either of the images, a penalty is added to the matching cost.

4.4.2 Stereo Triangulation of Vertices

The 3D geometry of an object is obtained by range computation from (at least) two images with different views. Among the many existing methodologies, the vertex triangulation is the most popular one [2, 33, 103]: with the camera model and a given world coordinate system as the common reference, it infers the 3D location of a remote point from a pair of 2D projective views at two camera positions.

The representation of camera pose adopted here is the combination of a 3×3 rotation matrix R and the 3×1 translation vector $T = \{T_x, T_y, T_z\}$ into a single 3×4 homogeneous matrix H:

$$H = \begin{pmatrix} R_{00} & R_{01} & R_{02} & T_x \\ R_{10} & R_{11} & R_{12} & T_y \\ R_{20} & R_{21} & R_{22} & T_z \end{pmatrix} \tag{4.11}$$

Figure 4.7 illustrates the geometrical configurations in the range computation. Suppose that in a pair of images I_a and I_b, the projections of the same object point P_w are $(u_a, v_a)^T$ and $(u_b, v_b)^T$, and the homogeneous matrices for the two cameras are A and B respectively. If the camera's focal length f is known and the origin of a camera coordinate system is chosen to be the center of the lens (refer to Figure 4.1), from simple geometrical computation, the coordinates of the image points p_a and p_b in the two camera coordinate systems are:

$$p_a = [f \cdot u_a, \ \ f \cdot v_a, \ \ f]^T, \qquad p_b = [f \cdot u_b, \ \ f \cdot v_b, \ \ f]^T.$$

Suppose that in the world coordinate system, the coordinate of the object point is (X_w, Y_w, Z_w), then with the transformation equation between the world system

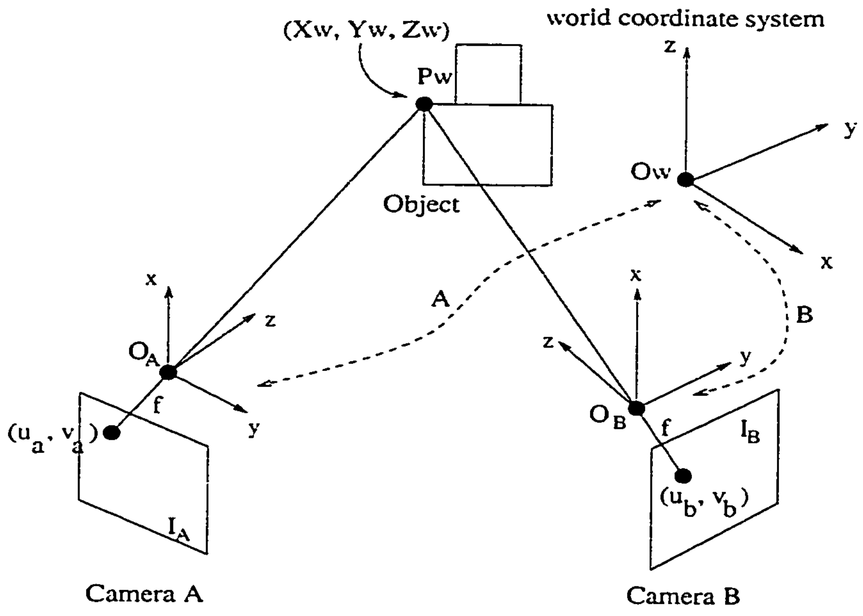


Figure 4.7: Range computation by vertex triangulation on stereo images.

and the camera systems we have:

$$\begin{pmatrix} f \cdot u_a \\ f \cdot v_a \\ f \end{pmatrix} = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{4.12}$$

$$\begin{pmatrix} f \cdot u_b \\ f \cdot v_b \\ f \end{pmatrix} = \begin{pmatrix} B_{00} & B_{01} & B_{02} & B_{03} \\ B_{10} & B_{11} & B_{12} & B_{13} \\ B_{20} & B_{21} & B_{22} & B_{23} \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{4.13}$$

By combining the above two equations, we obtain a set of linear equation from which we can calculate (X_w, Y_w, Z_w):

$$\begin{pmatrix} u_aA_{20} - A_{00} & u_aA_{21} - A_{01} & u_aA_{22} - A_{02} \\ v_aA_{20} - A_{10} & v_aA_{21} - A_{11} & v_aA_{22} - A_{12} \\ u_bB_{20} - B_{00} & u_bB_{21} - B_{01} & u_bB_{22} - B_{02} \\ v_bB_{20} - B_{10} & v_bB_{21} - B_{11} & v_bB_{22} - B_{12} \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} = \begin{pmatrix} A_{03} - u_aA_{23} \\ A_{13} - v_aA_{23} \\ B_{03} - u_bB_{23} \\ B_{13} - v_bB_{23} \end{pmatrix} \tag{4.14}$$

In fact, three equations of the above four are sufficient to compute $(X_w, Y_w, Z_w)^T$. Since each equation represents a plane, theoretically, all four planes intersect at one point. However, due to the errors, the four possible solutions from the combination of three of four equations may not be identical. It is preferable to employ the least square method to make use of all the four equations for solving $(X_w, Y_w, Z_w)^T$.

4.4.3 Shape Synthesis of Ellipse

Through vertex triangulation, the 3D geometry with polygonal features is recovered. The same approach cannot be applied to 3D circles since there is no simple way to locate and match a pair of control points on the 2D projections of the circle (which are ellipses). Thus, even if two ellipses are successfully extracted, fitted and matched in two images, its 3D shape cannot be calculated through simple vertex triangulation.

Projective inversion (also called perspective inversion) of ellipses in 2D images is a methodology to infer partial information on 3D circles. If the radius of a circle is known, it attempts to recover the 3D orientation of the circle, starting with just *one* 2D projection on the image plane [36]. This methodology, combined with vertex triangulation, can be used to recover a 3D circle from two of its projections.

Projective Inversion of Ellipses

Projective inversion is mostly used for model based object recognition and pose determination. In 2D images, the circular parts of the cylindrical features are projected as partial or complete ellipses. If the radius of the circle in the model base is known, its 3D orientation can be calculated [36, 45]. However, different from these problems, the radius of the circle is unknown in model synthesis. Thus, the actual 3D measure of a circle cannot be obtained from a single view. Nevertheless, with multiple views of the same circle and the corresponding camera poses, it is possible to recover the 3D circle by *combining projective inversion and vertex triangulation.*

In general, to determine the equation of a circle in 3D space, we need at least three points on the circle. In practice, it is very difficult to establish correspondence for points *on* a circle or ellipse. What can be calculated are the following parameters: a) the length of the long axis, b) the length of the short axis, c) the center and d) the tilting angle. With two images containing the projections of a same circle, the following steps can recover the 3D circle:

1. arbitrarily assign a radius of the 3D circle to be recovered;
2. use projective inversion of ellipse, with the assigned radius and the known camera poses, reconstruct a circle *hypothesis* in 3D space for each of the images;
3. according to the known camera poses, project the center and the virtual normal point of the hypothetical 3D circle onto the 2D planes of each image (the virtual normal point is defined as the point along the normal vector of the circle whose distance to the circle center is the radius);

4. do the vertex triangulation of the center and the virtual normal point respectively and then find the true radius of the 3D circle;
5. use the calculated true radius to do the ellipse inversion again to find the orientation of the circle.

The arbitrarily assigned radius at the first stage is only used to find the projections of the circle's center and the normal point, and then to establish the matching ellipse pair. Vertex triangulation will reveal more accurate radius and orientation of the circle finally.

An Equivalent Problem

In order to elaborate the procedure to obtain a closed form solution for projective inversion of an ellipse, we first define an equivalent problem.

As shown in Figure 4.8, the problem of inverting the ellipse a in the 2D projective plane Π_1 to a 3D circle A is equivalent to determining the plane Π_2 whose intersection with the cone C (with the vertex at the optical center O and having an elliptical intersection a over the image plane Π_1) forms the circle A. Given a desired radius value r of the circle A to be solved, we are able to determine up to two candidate solutions.

Closed Form Solution

As shown in Figure 4.8, we choose the x_1-y_1-z_1 coordinate system such that: (1) the optical center is at the origin O; (2) axis z_1 is coincident with the optical axis: and (3) axes x_1 and y_1 are parallel to X and Y in the image system respectively.

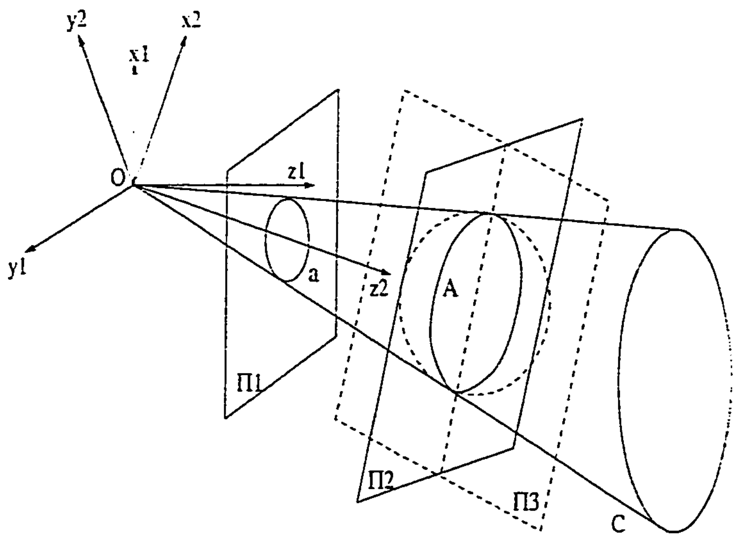


Figure 4.8: Reference frames. planes and coordinate systems in projective inversion of ellipse.

For an image plane Π_1 normal to the z_1-axis where $z_1 = f$, the equation for the observed ellipse on the image plane is in the following form:

$$m_1x_1^2 + m_2x_1y_1 + m_3y_1^2 + m_4x_1 + m_5y_1 + m_6 = 0 \tag{4.15}$$

The equation for the corresponding elliptic cone is:

$$m_1x_1^2 + m_2x_1y_1 + m_3y_1^2 + \frac{m_4}{f}x_1z_1 + \frac{m_5}{f}y_1z_1 + \frac{m_6}{f^2}z_1^2 = 0 \tag{4.16}$$

In matrix form:

$$(x_1\ z_1\ y_1)M\begin{pmatrix} x_1 \\ z_1 \\ y_1 \end{pmatrix} = 0 \tag{4.17}$$

where

$$M = \begin{bmatrix} m_1 & \frac{1}{2}(\frac{m_4}{f}) & \frac{1}{2}m_2 \\ \frac{1}{2}(\frac{m_4}{f}) & \frac{m_6}{f^2} & \frac{1}{2}(\frac{m_5}{f}) \\ \frac{1}{2}m_2 & \frac{1}{2}(\frac{m_5}{f}) & m_3 \end{bmatrix} \tag{4.18}$$

For simplification, a transformation T is performed to change the reference frame such that z_2 is in the direction of the elliptic cone:

$$\begin{pmatrix} x_1 \\ z_1 \\ y_1 \end{pmatrix} = T \begin{pmatrix} x_2 \\ z_2 \\ y_2 \end{pmatrix} \tag{4.19}$$

This is an orthogonal transformation so $T^{-1} = T'$. Then Equation 4.17 becomes

$$(x_2\ z_2\ y_2)T'MT \begin{pmatrix} x_2 \\ z_2 \\ y_2 \end{pmatrix} = 0 \tag{4.20}$$

$\Rightarrow$

$$\lambda_1 x_2^2 + \lambda_2 z_2^2 + \lambda_3 y_2^2 = 0 \tag{4.21}$$

where λ_i are eigenvalues of $\tilde{M} = T'MT$, and $\lambda_1 > 0$, $\lambda_3 > 0$, $\lambda_2 < 0$.

As shown in Figure 4.8, the plane Π_3 which contains the original circle is a rotation of the plane Π_2 containing the elliptic projection of the circle, centered on the axis z_2, distanced f' from the origin, and parallel to the $x_2 - y_2$ *plane*. Let x_3 be parallel to Π_3 and z_3 be in the direction of the normal to Π_3.

$$x_3 = x_2 \cos\alpha + (y_2 - f')\sin\alpha \tag{4.22}$$

$$z_3 = -x_2 \sin\alpha + (y_2 - f')\cos\alpha \tag{4.23}$$

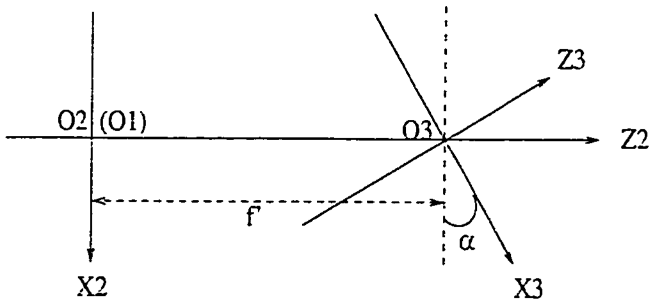


Figure 4.9: Top view of the x2-z2 plane.

$$y_3 = y_2 \tag{4.24}$$

Then, with a given radius r, the center of the 3D circle is:

$$\begin{cases} x_3^* &= r\sqrt{\frac{-\lambda_2}{\lambda_1}}\sqrt{\frac{\lambda_1-\lambda_3}{\lambda_3-\lambda_2}}\sin\alpha \\ z_3^* &= 0 \\ y_3^* &= 0 \end{cases}$$

where $\sin\alpha = \sqrt{\frac{\lambda_1-\lambda_3}{\lambda_1-\lambda_2}}$. and the equation of plane Π_3 is simply $z_3 = 0$. The location of the center and the plane equation in the original coordinate system can then be easily figured out. A detail derivation can be found in [36].

4.4.4 Partial Model Construction

Face Model

From the 3D circles, points and edges (that connecting the points), the most straightforward method to represent a 3D object is the use of a *wire-frame*. A wire-frame is a 3D object model consisting of the surface discontinuity edges of the object. Some of the earliest computer vision systems [75] used wire-frame models to represent polyhedral objects. These models continue to play the largest role in the current 3D vision systems.

In a wire-frame model, there are two types of elements: lines and nodes. The lines correspond to the characteristic edges on the surface boundary, and the nodes are the connecting/intersecting points of the edges. For simple polyhedral objects, the wire-frame representation can be precise.

Considering we are to build a triangular mesh as the bridge connecting descriptive shape modeling and symbolic AHR, a more generic dual of wire-frame model is the face modeling based on the faces of the objects. In a face model, the elements are the surface patches bounded by closed circuits of wire-frame edges. The collection of such faces forms a complete 3D surface boundary of the object [60]. Using faces to model an object has several advantages over wire-frames:

- fewer faces than corners/edges are required to describe the same object;
- a face description captures more significant information on geometry;
- in a face model, a surface patch can be bounded by any type of closed curves in addition to the wire-frame edges, which makes it straightforward to add detected 3D circles in the model.

The procedure to extract faces from wire-frames is adopted from [29] (refer to Figure 4.10)). First, a procedure called *Tablet_Corner_Edge*() handles the corners and edges resulting from vertex triangulation. An incidence table is generated as shown in Table 4.2. Empty columns, i.e., "lonely" corners, are considered as noise and are removed from the table.

Merge_Corners() and *Merge_Edges*() are the procedures to eliminate duplicated edges/corners due to noisy input data (e.g., the reflections around a sharp

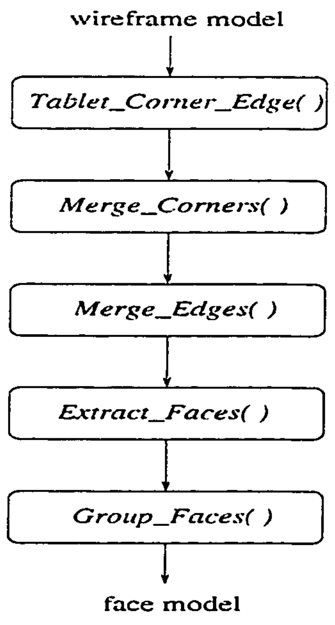


Figure 4.10: The process to construct face representation from wire-frame model.

Table 4.2: The incidence table of corners and edges.

	C_1	C_2	$\cdots$	C_n
E_1	1	1	$\cdots$	0
E_2	0	1	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
E_m	0	0	$\cdots$	0

edge). $Merge\_Corners()$ scans through the set of 3D corners. The Euclidean distance between pairs of neighboring corners is evaluated. If the distance value is less than a preset threshold (normally the distance threshold is about 2-3 pixels in length if projected to the 2D images), the two corners are merged into one. On the other hand, $Merge\_Edges()$ scans through the set of edges. The smallest distance and the angle between two edges are evaluated. If two edges are close to each other and the angle between them is very close to 0 or π, the two edges are merged into one. According to the corner and edge merging, the incidence table is adjusted. Then the faces are extracted by procedure $Extract\_Faces()$. A face is a set of edges forming a closed cycle. Each face is represented by:

1. a set of edges or curves to form the face boundary;
2. a set of optional corners;
3. a vector signifing the normal direction of the face.

The edge-corner incidence matrix is a representation of a connected graph which is equivalent to the wire-frame model. Each face of the object corresponds to a set of edges forming a closed loop. Searching for a closed cycle in a graph is a common task in linear graph theory. Here a depth-first search algorithm is applied. Different from the conventional cycle searching algorithm, in the face searching, there is an additional constraint to judge if a cycle consists of a face of the object: all the edges in the cycle have to be approximately on the same plane. So the search algorithm begins with a set of three connected corners, from which an approximated normal vector of the possible face can be calculated. During the expansion of the cycle, this normal vector is consistently updated according to the new corners/edges added. The face grouping procedure has the following steps:

1. set the face set S_f empty;
2. find all ordered lists of 3 corners in which the 3 corners are connected consequently by 2 edges, denoted as the list $L_i = \{c_{i_1}, c_{i_2}, c_{i_3}\}$ with corners c_{i_1}, c_{i_2} and c_{i_3}, where $1 \leq i_1, i_2, i_3 \leq n$, n is the total number of corners, $1 \leq i \leq k$ is the index of list, and k is the total number of 3-corner lists;
3. let $i = 1$;
4. for each list L_i, calculate the normal vector of the plane defined by all the corner points in the list, denoted as n_i;
5. if the tail of L_i is connected to its head, then list L_i represents a face; put L_i in S_f, and go to step 8;
6. if there exists a corner c_j in the corner set that satisfies:
   - c_j is not in L_i yet;
   - c_j connects the current tail of L_i;
   - c_j is approximately in the plane defined by all the corners in L_i;

   then put c_j in L_i as the new tail; if no corner satisfies the above conditions, go to step 8;
7. go to step 4;
8. let $i = i + 1$, then if $i \leq k$ then go to step 4;
9. output S_f and exit.

Finally, procedure *Group_Faces*() groups the faces together if they have one or more common edges and their normal vectors are collinear/parallel.

Triangular Mesh Model

Building a triangular mesh is straightforward from the face model. The faces are first converted into the form of polygons or polygonalized circles. The following recursive procedure decomposes a polygon P_0 with n edges and n corners into $n-2$ triangles:

1. set the triangle list T empty, let $m = n$;
2. let $j = 0$;
3. in polygon P_j, start from any point, label the corners from 1 to m consequently, denoted as c_1, c_2, ..., c_m, where m is the number of corners in polygon P_j;
4. let $i = 1$;
5. add an edge from corner i to corner $i+2$ to make up a new triangle $t = \{c_i, c_{i+1}, c_{i+2}\}$, and add t to T (if $i+2 = m+1$, then use c_1 as c_{i+2});
6. let $i = i + 2$;
7. if $i < m$, go to step 5;
8. use the nodes c_1, c_3, ..., c_{2k+1}, ..., and the newly added edges to construct a new polygon P_{j+1} with reduced numbers of corners and edges;
9. if P_{j+1} is a triangle, then add P_{j+1} to T, output T and exit;
10. let $j = j + 1$, go to step 2.

Since the triangles generated by the above processes are directly from the face models, they automatically satisfy the conditions of a triangular mesh described in

Section 3.3.2. Then, an attributed hypergraph (AH) that represents the 3D information corresponding to the face model can be obtained by applying the procedures described in Section 3.3.2.

4.5 Augmented Reality Construction with AHR

To build an augmented scene, virtual objects/scenes have to be combined with the the representations of real objects/scenes constructed by the process described in the previous sections. Up to now, range data have been sensed, and modeled in the form of several AH's, each of which represents a partial 3D scene (called a patch). In the following, these AH's are incrementally synthesized into an integrated AHR, which is then augmented with the AHR's representing virtual objects/scenes.

The *augmentation* here stands for not only integrating different AHR's, but also supplementing the AHR's with entities that may appear neither in the real scenes nor in the virtual scenes. With the unified AHR, object properties such as surface colors and textures are represented as attributes, which makes it easy for us to perform the integrating and supplementing tasks required by most AR applications.

4.5.1 AHR Synthesis

The basic synthesis algorithm works for two AHR's. With more than two AHR's, the algorithm works in an incremental way such that the synthesis is conducted between each pair of AHR's that are consequent in the AHR sequence.

Synthesis of Two Attributed Hypergraphs

Given two attributed hypergraphs $G_1 = (X_1, Y_1)$ and $G_2 = (X_2, Y_2)$, the synthesis process is organized into a two-level process to obtain the attributed hypergraph $G_r = (X_r, Y_r)$.

At the first processing level, the two sets of hyperedges Y_1 and Y_2 are considered. Let us consider $e_p \in Y_1$ and $e_q \in Y_2$. If by comparing their attribute values, e_p and e_q correspond to the surface patches on the same plane, and they touch, contain or overlap with each other, then they should be integrated into one patch by averaging their attribute values and composing the adjacency properties. For each hyperedge in G_1 the comparison is performed to search for its counterpart in G_2. If found, the two are combined and passed to the next level for elementary graph synthesis, after which the attribute values and adjacencies of the combined hyperedge are re-computed and put into Y_r. For a hyperedge either in G_1 or in G_2 with no counterpart found in the other hypergraph, they are directly transferred into Y_r by a union operation.

The second processing level refines the elementary graph in the DAH after the patch synthesis. The elementary graph corresponds to the triangular mesh constructed from the sensory data. There may be triangles that are identical, containing, being contained, or overlapping. In such cases, the vertices and edges have to be re-calculated, otherwise, the vertices or edges are considered as new ones and simply copied to X_r. The adjustment of the vertices has several possibilities:

- if the two triangles to be compared are identical, the vertices of either one of the two is copied to X_r;
- if one of the two triangles is found to contain (or to be contained by) the

other by comparing the attribute values, then the vertices of the former (or the latter) is copied to X_r and the latter (or the former) is ignored;

- if the two triangles are overlapping, then:
    1. a one-to-one correspondences between the three triangle points are established between the two triangles;
    2. the average point for each pair of matched triangle points is calculated;
    3. an average triangle is constructed, whose triangle points are the average points;
    4. the three vertices of the average triangle are copied to X_r.
- update the edges and then the hyperedges in Y_r according to the new X_r set.

Incremental AHR Synthesis

The above synthesis process considers only one pair of attributed hypergraphs. With multiple views, such synthesis could be performed many times and a final model is constructed in an incremental way.

Suppose that we have n range models in the form of attributed hypergraphs, denoted as $G_1, G_2, ..., G_n$. Then we can conduct the synthesis for $n-1$ times on each pair of the consequent images in the sequence, during which a 3D AHR is incrementally constructed. The possibly redundant information in these partial AHR's obtained from different views enables us to perform the following improvements on the recovered 3D data:

- *Compensate for system errors*

  System errors are brought in by the digitization error, feature detection error, computation error in vertex triangulation, etc. The errors can be compensated by averaging the resulting range data.

- *Recover missing features*
  Missing object parts in a range image can be caused by occlusions, unrealistic illuminations, or errors in feature detection. However, they still have good chances to be present in other range images.

- *Eliminate erroneous edges and corners*
  Noise on the object's surfaces and in the background may form erroneous corners or edges. In incremental AHR synthesis, a score representing the number of appearance for each vertex is added. Since the noise has very little chance of being repeated at the same place, it will have a very low score. At the final stage, vertices with low scores can be eliminated.

The incremental AHR synthesis consists of the following steps:

1. let $i = 1$, set AHR G_a empty;

2. match the two attributed hypergraph G_i and G_{i+1} and obtain a synthesized G_{r_i};

3. compare the G_{r_i} with G_a:

   (a) if a new vertex's geometrical attributed value is close enough to one that is already in G_a, they are considered the same vertex. The average between them replaces the old one and the score for this vertex is increased by 1;

   (b) if no vertex in G_a is close to the new one, add the new vertex to G_a and set its score as 1;

   (c) adjust the edges and hyperedges that contain the modified vertex;

4. let $i = i + 1$;

5. if $i < n$, go to step 2.

6. eliminate the vertices in G_a with scores lower than the preset threshold;

7. adjust edges and hyperedges of G_a after noisy vertex elimination.

The result of the synthesized 3D scene is an attributed hypergraph, which at this moment consists of the geometrical properties of the scene synthesized from the vision sensor data. In the following sections, we will discuss the integration of virtual objects to form an augmented scene based on the AHR.

4.5.2 AHR Augmentation

At the representation level, the augmentation of a 3D scene is equivalent to the augmentation of the corresponding AHR, which can be performed by applying a proper set of AH operators. Typical augmentations in a constructed scene may include:

- *integration with virtual scene*
  A typical AR world normally consists of objects constructed from the real 3D scene and additional objects or backgrounds imported from the virtual world. One of the most important problems in integration is the proper alignment of the virtual parts and the real parts.

- *change of surface colors and/or textures*
  The objects in the AR world may change their surface colors and/or textures

to present certain effects (for example, shadows and color changes due to the virtual light sources);

- *shape changes by deformations*
  When the AR world is presented for physical simulation, it is possible that the objects, either from real views or from virtual world, have shape changes due to the physical interactions among them.

As we can see from the above possibilities, when augmentation is performed on AHR of a 3D scene, there may be qualitative changes, such as emerging parts or deformations, or quantitative changes on the attributes, such as the changes of locations, colors and textures. The operators defined in Section 3.3.3 facilitate such flexibility for AR constructions with AHR. In the following, the above three aspects to construct the augmented world with AHR are discussed.

Virtual Objects Integration

At the AHR level, the integration of a virtual object with the 3D scene constructed from sensory data (or vice versa) is the task of the combination of two or more AH's. In Definition 3.16 and Definition 3.17, we have defined the operators $\bigcup$ (*union*) and $\bigcap$ (*intersection*) on two AH's, which can be directly applied here.

Suppose that the AHR of the real scene is G_r and the AHR of the virtual part is G_v, the combined AH G is obtained by:

1. calculate $G_u = G_r \bigcup G_v$ and $G_i = G_r \bigcap G_v$;
2. let $G_o = G_u \setminus G_i$;
3. align the attributes of all entities in G_i;

4. set $G = G_o \bigcup G_i$.

The "alignment" of the attributes of two AH's depends upon the application context. Normally, it only involves translations and/or rotations on the AH's to properly position the objects in the AR scene. Thus, one or more *attribute transitions* will handle it. However, in some cases, there may be qualitative changes of the AH's associated with the integration. In the fusion of two 3D objects which may be required by AR tasks, the integration of the two AH's that represent the two objects, denoted as G_r and G_v, has to be followed by eliminating all redundancies (e.g., inner surfaces or inner vertices) to ensure a fully integrated shape. The following steps will remove the inner vertices and surfaces:

1. calculate G which is the integration of G_r and G_v (with possible inner vertices and/or surfaces), and $G_i = G_r \bigcap G_v$;
2. if a vertex $v_k \in G_i$, v_k is an inner vertex:
   (a) find a vertex $v_l \in G$ such that v_l is adjacent to v_k and the discrepancy between v_l's attribute value Av_l to v_k's attribute value Av_k is the smallest among all vertices adjacent to v_k;
   (b) in G, apply *merge* on v_k and v_l;
3. if a hyperedge $H_k \in G_i$, H_k becomes an inner hyperedge:
   (a) find a hyperedge $H_l \in G$ such that H_l is adjacent to H_k and the discrepancy between H_l's attribute value Ae_l to H_k's attribute value Ae_k is the smallest among all hyperedges adjacent to H_k;
   (b) in G, apply *join* on H_k and H_l;
4. if there is no inner vertex or hyperedge, exit, otherwise go to step 2.

Texture and Color Mapping

Natural looking surface textures (or surface colors when texture is not considered) are of great importance to the construction of augmented reality. To meet this requirement, a texturing or coloring algorithm is adopted to estimate the surface appearance for each 3D triangle in the mesh from the available 2D views. More details on the adopted algorithm are given in [68].

Figure 4.11 illustrates the principle of binding surface appearance information. With calibrated camera parameters, the projective relation between the 3D shape and their 2D views in the 2D images can be established in the form of a set of equations (refer to Equations 4.12 and 4.13). Then for each triangular surface in the reconstructed 3D model, the 2D image with the largest viewing area of the triangle is selected. The texture and/or color of the projected triangle in the selected 2D image is inverted into 3D space and clipped on the corresponding 3D triangle.

The surface texture (or color) binding algorithm has the following modules:

1. *3D texture (or color) region grouping*

   Since the object is represented by a triangular mesh, smoothly connected triangles with homogeneous textures (or colors) in their 2D views are grouped together to reduce the computational cost [101]. Local geometrical features are also considered as constraints to make sure the triangles in the same texture (color) group are smoothly connected. A closed convex region that has invariant normal vector is preferred for binding to reduce the total length of boundary.

2. *2D projection selection*

   In most cases, a grouped region is visible in more than one of the images. It

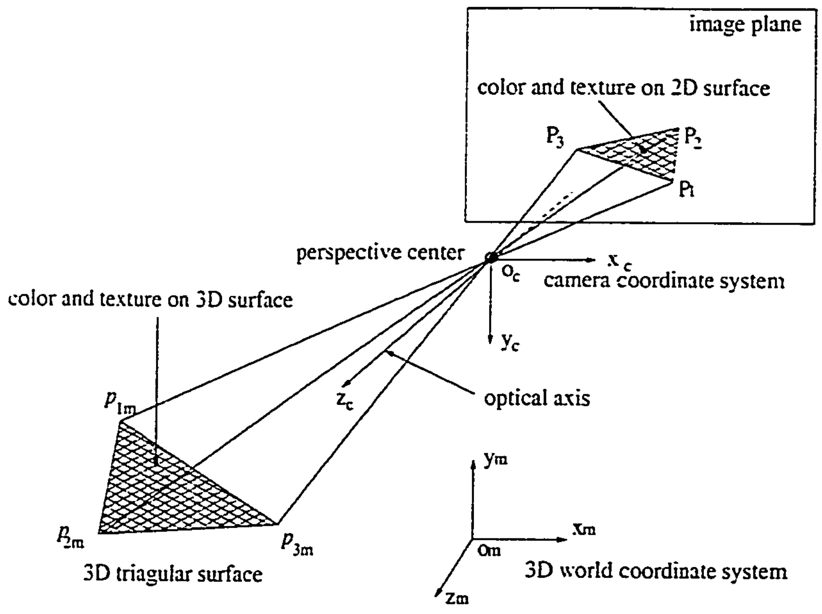


Figure 4.11: The principle of texture inversion from 2D image to 3D surface.

is reasonable to assume that the more exposure of the texture region in an image, the more accuracy will be achieved when the region is inverted into 3D space. Therefore, the image with the largest viewing area of the grouped texture region is selected and a 2D to 3D inversion is performed with the camera pose associated with this image.

3. *Boundary processing*

   Special care has to be taken along the boundaries between different texture regions. A simple averaging filter is added for the local areas around the boundaries to blend the texture. Although this results in a local blurring effect, it is more visually acceptable than the abrupt change without the filter, which is called the "blocking effect".

4. *Texture synthesis for occluded region*

   It is possible that the available views do not cover all the surfaces of the 3D object/scene. For color attributes, a simple duplication of the color from the nearby triangles is adequate. If textures are processed, the texture of the invisible regions are synthesized from visible regions depending on the view point. The synthesized textures or duplicated colors may not correspond to the real ones but are still less conspicuous than texture/color "holes".

Shape Deformations

The problem of shape deformation arises when the objects/scenes are modeled in AHR with physical parameters as attribute values. Physical features as attributes in AHR describe deformable objects by (1) the physical properties of the objects; (2) physical state transitions; (3) initial states and (4) external factors such as forces.

In an augmented scene, the augmentation may include the changes of physical states for existing objects or virtual objects, possibly with additional external forces. In the above cases, due to the physical interactions among the objects, there may be deformations. Therefore, a recalculation of the physical states of the objects is required. More details on the calculations of shape deformation can be found in Section 5.3.

4.6 Experiments

An AHR based 3D modeling system has been implemented as a research prototype in the PAMI Laboratory of University of Waterloo. It is programmed with OpenGL and Motif libraries on an SGI Indy Workstation (RISC4000). Experiments have been conducted on several indoor scenes captured by CCD cameras. In the following, the experiments are separated and discussed in two sections according to the functional modules applied in AR construction:

1. Section 4.6.1 shows the experiments that apply computer vision technologies to perform 3D model synthesis with CCD cameras;
2. Section 4.6.2 demonstrates the AHR construction and augmentation over the geometrical models synthesized in Section 4.6.1.

4.6.1 Vision Based 3D Geometry Recovery

CAD Model Synthesis of A Simple 3D Object

The first experiment of model synthesis was carried out on a simple aluminum object which could be used for camera calibration. The object to be modeled is placed on a dark, flat surface as the testing base within the stretching range of a PUMA 760 robot arm. When the robot arm is rotated around the object, images are taken by a CCD camera mounted on the last link of the robot from different vantage points.

The object to be tested is called a "bridge" model. Twelve images of the object are taken from different views under normal lighting condition. For each image acquired, the camera pose is calculated from the reading of the robot end-effector. 2D visual features, such as lines, corners and ellipses are extracted as described in Section 4.2. Figures 4.12 (a) through (f) show six of its twelve views with feature detection results, and Figures 4.13 (a) through (d) illustrate partial model synthesis results from the images acquired at the second, fifth, seventh and the final views respectively. The incrementally synthesized model is depicted by the thick white

Table 4.3: The mean error (measured in pixel) of the synthesis model "bridge" in incremental model synthesis.

view # (m)	1	2	3	4	5	6	7	8
n	8	9	11	11	11	13	13	13
e_n	0.5840	0.5375	0.4535	0.4145	0.4013	0.3606	0.3270	0.3238

lines which are the back-projection of the 3D CAD model onto the images. In the last image (Figure 4.13 (d)), a complete model of the "bridge" is constructed.

The actual object was manually measured and then compared with the model synthesis results. The algorithm provided in [3] is applied to compute the R and T between the actual model and the synthesized one. Suppose that the feature point set in the actual model is $\{p_1, p_2, ..., p_n\}$ and the corresponding synthesized feature point set is $\{p_1', p_2', ..., p_n'\}$; it has been proved in [3] that the computed R and T will minimize the overall error $E_n = \sum_{i=1}^{n} ||p_i' - (Rp_i + T)||$. The average error per feature point $e_n = \frac{E_n}{n}$ represents the non-rigid deformation between the two point sets, which can be used to measure the accuracy of the synthesized model. In the process of incremental model synthesis, as the frame number m increases, n is gradually growing since more and more images are taken. From Table 4.3 and Figure 4.14, as we expected, it can be seen that e_n decreases as n and m increase since more source images are available to generate the model.

3D Indoor Scene Reconstruction

A typical application of 3D scene reconstruction is *vision based autonomous rover navigation*: an autonomous rover is equipped with two cameras through which the

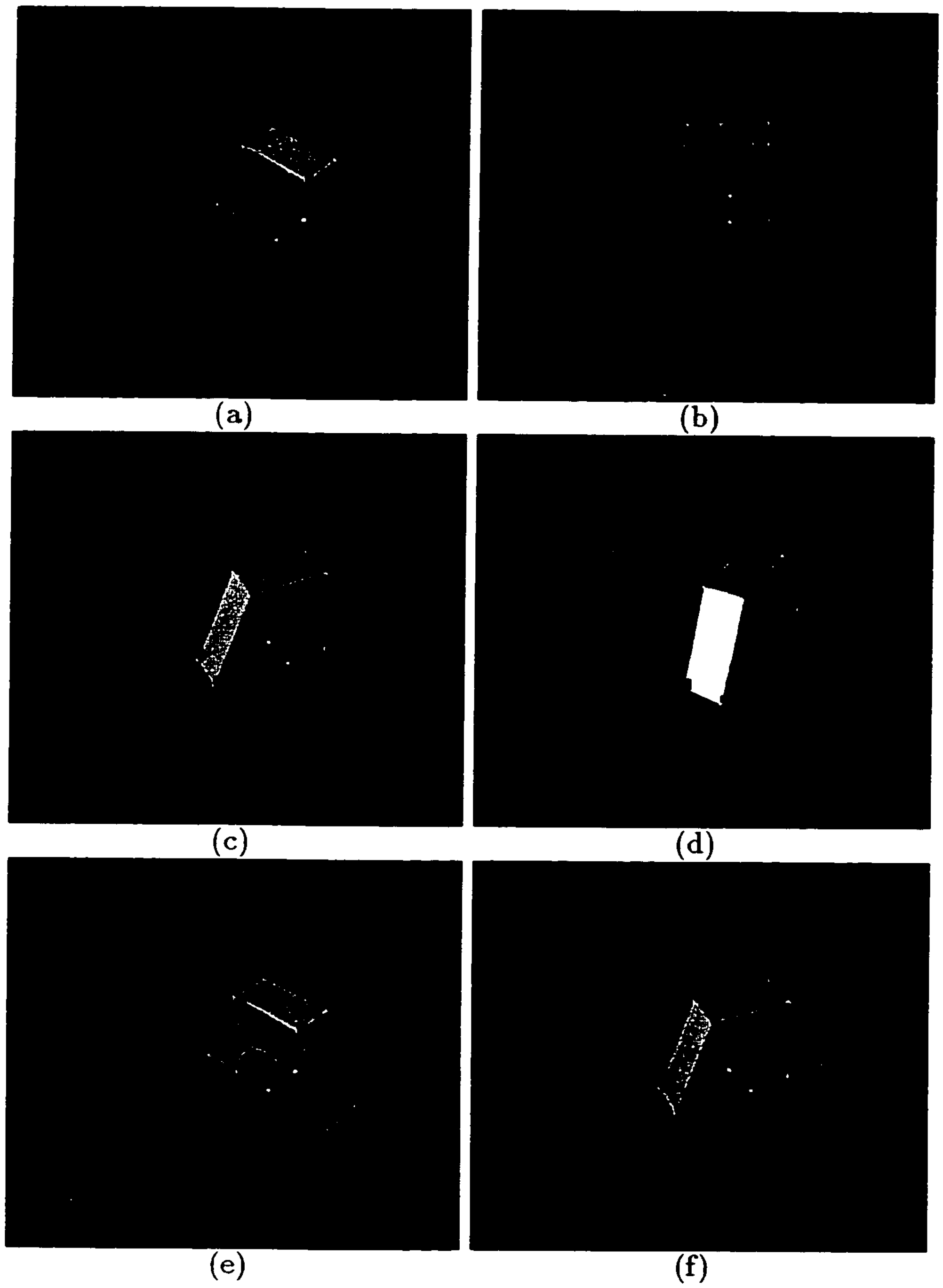
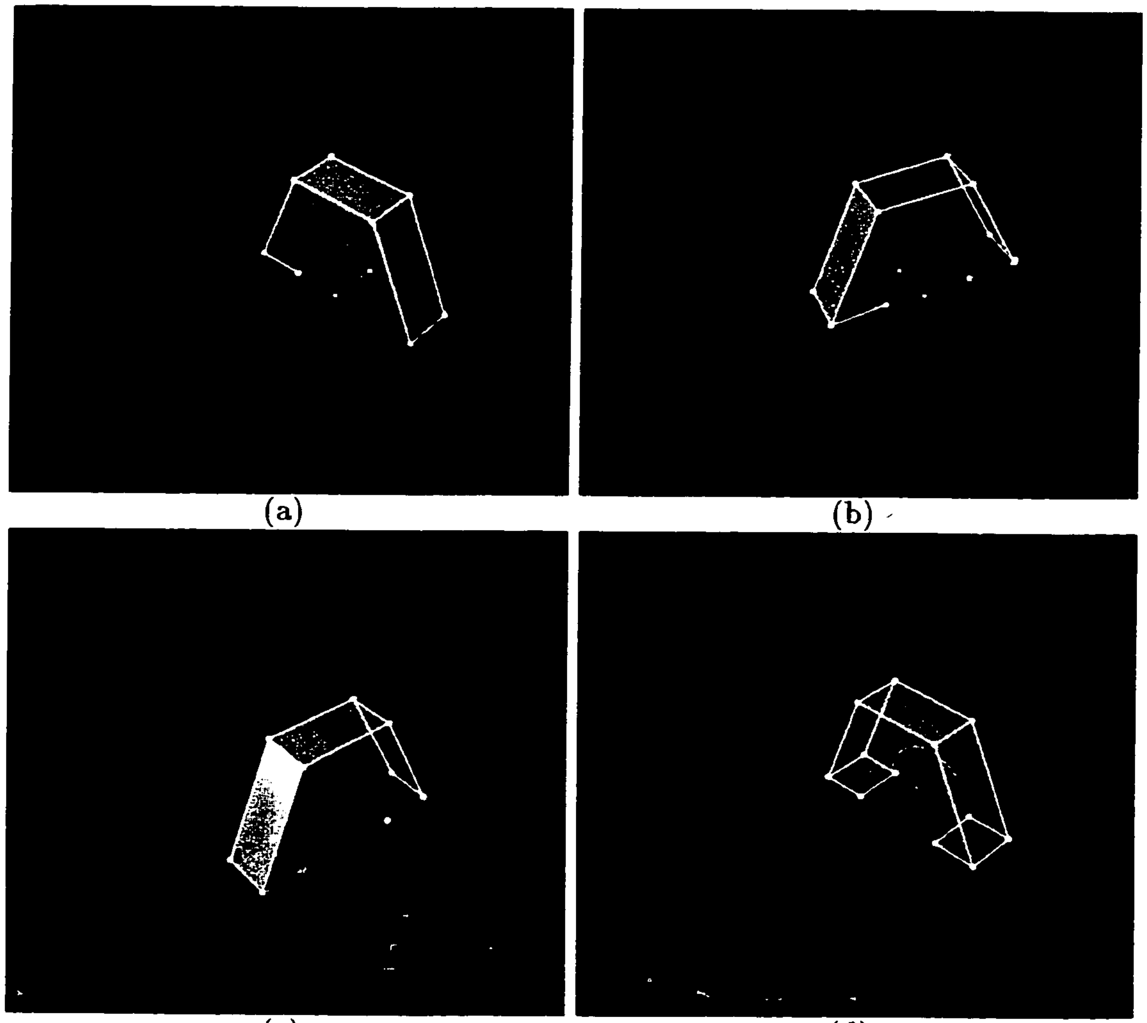


Figure 4.12: (a)-(f) The images and feature detection results of six different perspective views of the "bridge".



(a) (b) (c) (d)

Figure 4.13: (a)-(d) The model synthesis results for the "bridge" at view #2, view #5 view #7 and view #12 respectively.

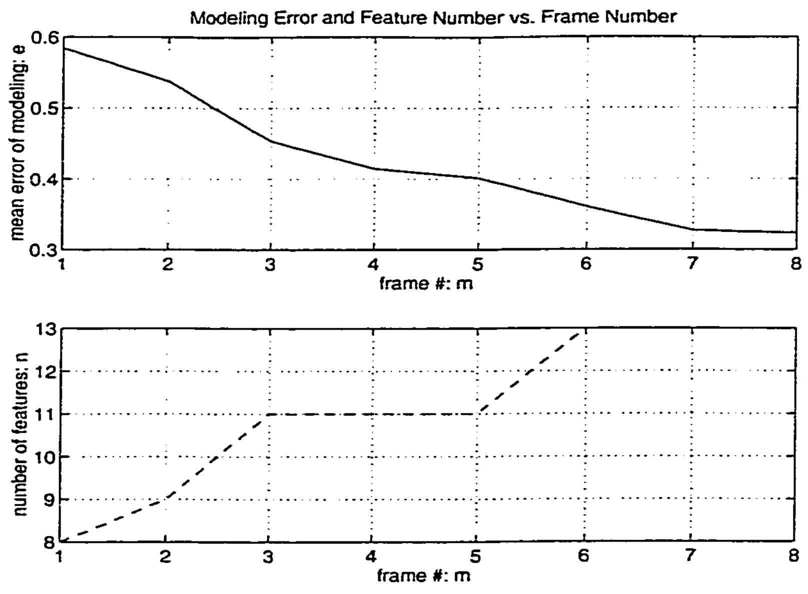


Figure 4.14: The modeling error and the number of extracted features vs. frame number in incremental model synthesis.

environment is modeled by an intelligent stereo vision system to guide and plan the rover's trajectory and/or task.

To simulate the stereo vision system in such an application, in the second experiment, a color CCD camera was put at two vantage points and two pictures of a table in a room were captured. Each picture was digitized by the digitizer equipped on the Silicon Graphics workstation. The two images are shown in Figures 4.15 (a) and (b). From the images, the raw features extracted are mainly corners and lines, as shown in Figures 4.16 (a) and (b). The corners of the table surface have been measured manually and are used as the landmarks. The 2D corner features obtained from the images are adequate for camera pose determination.

Different from the model synthesis of simple 3D objects placed on the testing base, in this experiment, due to the noise in the background and the complexity of the object's shape, limited supervision is required to pick up the salient 2D features

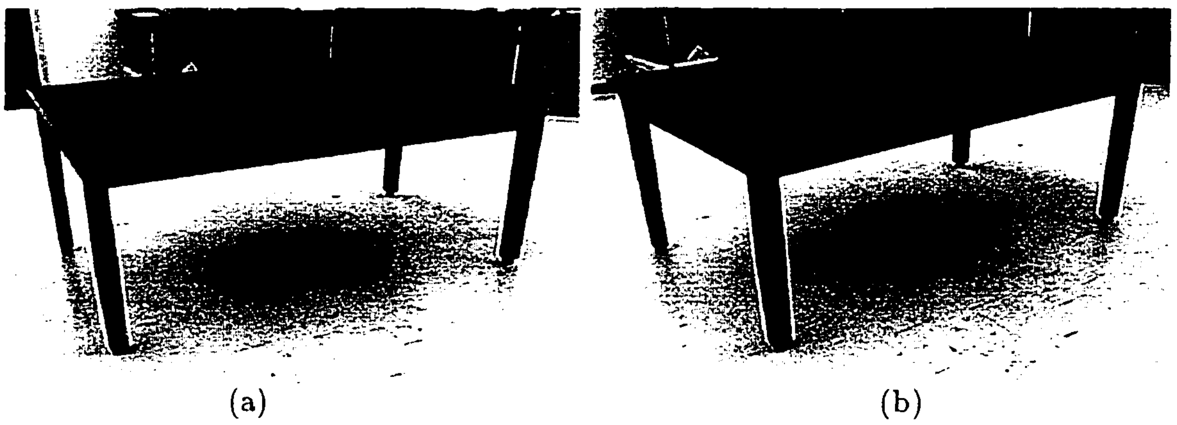
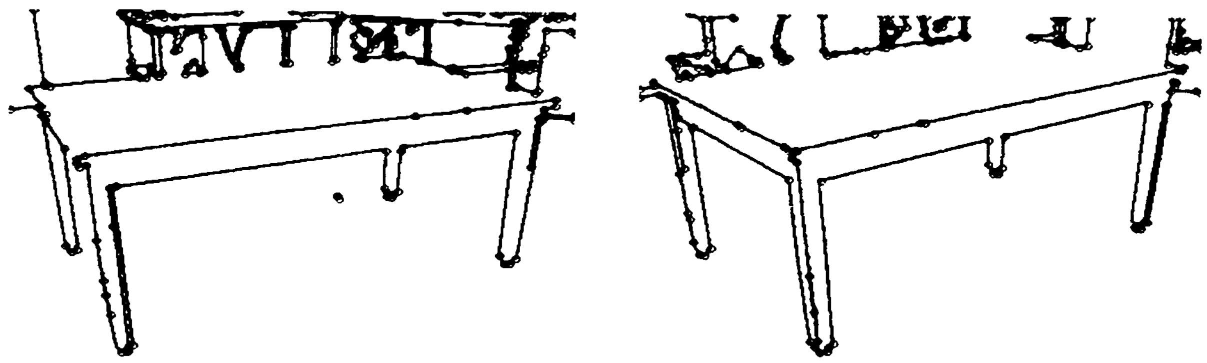


Figure 4.15: The images of the laboratory table captured by a CCD camera: (a) the right hand side image of the table; (b) the left hand side image of the table.

for vertex triangulation.

Figure 4.17 illustrates a partial table model constructed from the corner/line features back-projected to the left hand side image. It should be noted that some important points and lines in the model are occluded in the stereo views. They can be recovered if more perspective views are provided. However, in this experiment, they are picked up under simple user supervision. The model is not complete but is sufficient to build an AR or to be used as the landmark in other 3D reconstruction experiments for scenes containing this table.

Another experiment on indoor scene analysis involves the PAMI laboratory room. The left and right hand side images are shown in Figures 4.18 (a) and (b) respectively. Figure 4.19 gives the feature extraction results. The table appearing in both images serves as the landmark to determine the camera poses. Key features that are crucial to rebuild the room, such as the corners and lines formed by the intersections of the floor, the wall and the ceiling, are selected from the raw features to construct the partial CAD model of the room by the user. On the other hand,



(a) (b)

Figure 4.16: The feature extraction results: (a) the right hand side image of the table; (b) the left hand side image of the table.

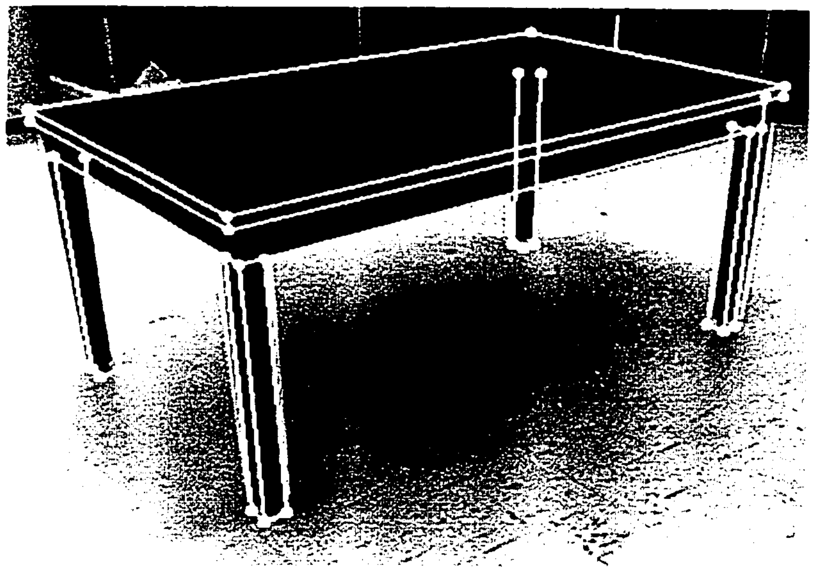


Figure 4.17: The CAD model of the laboratory table back-projected onto the left hand side image.

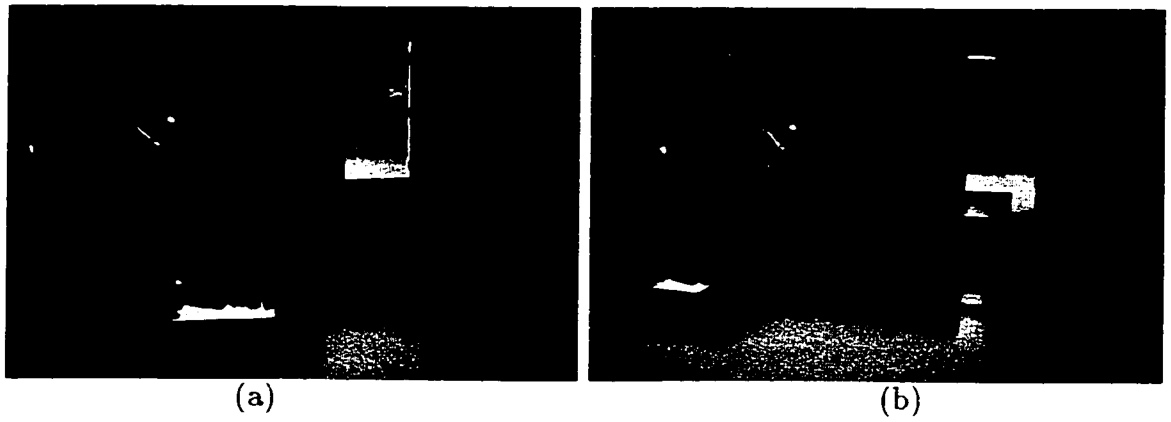


Figure 4.18: The images of the laboratory scene captured by a CCD camera: (a) the left hand side image; (b) the right hand side image.

detailed structures of the furniture in the corners and objects inside the door are ignored.

Figure 4.20 back-projects the constructed partial CAD model to the left hand side image. Since the model of the table in the scenes is known and has been used as the landmark to determine the relative camera poses, the partial table model can be replaced by its known full model as shown in the next section.

4.6.2 AHR and Augmented Reality

Laboratory Table

Figures 4.21 (a) though (f) demonstrate the experimental results on AR with the table modeled in the previous section. Figure 4.21 (a) depicts the result of automatic triangular mesh generation from the face model given in Figure 4.17. The corresponding AHR constructed from the mesh is illustrated in Figure 4.21 (b). In the figure of the AHR, the rectangles signify the triangles, whose colors represent

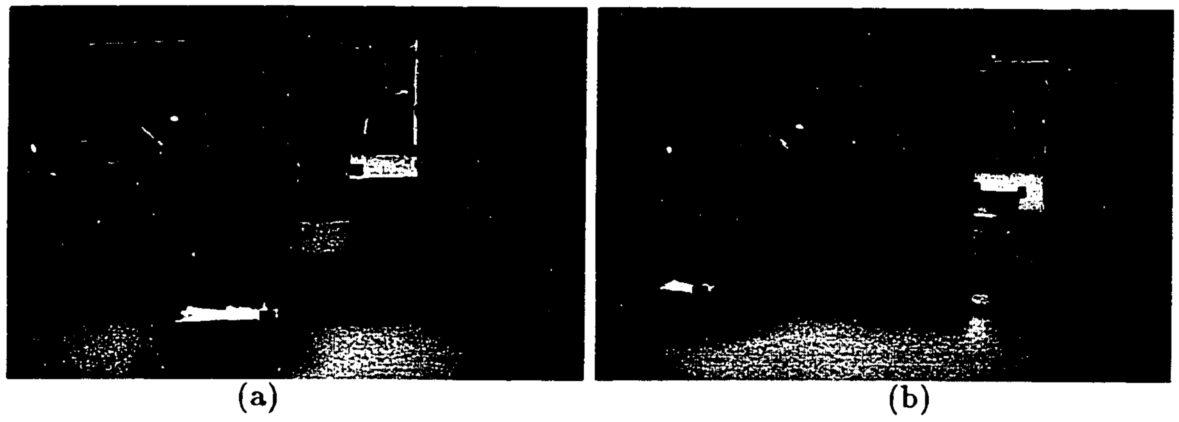


Figure 4.19: The feature extraction results on: (a) the left hand side image of the scene; (b) the right hand side image of the scene.

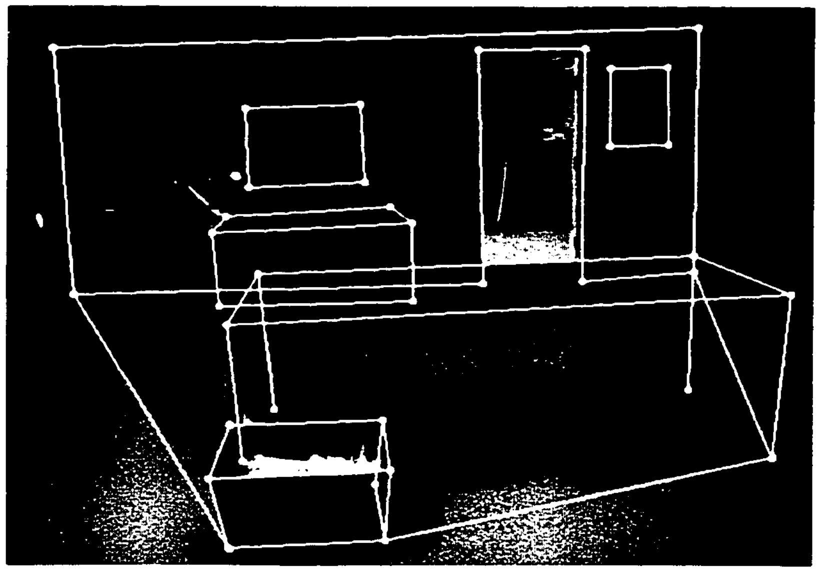


Figure 4.20: The rough CAD model of the laboratory scene back-projected to the left hand side image.

the orientations of the triangles. The links among the nodes denote the hyperedges:

- the nodes that are connected by links consist of a hyperedge, which is a meaningful part of the object;
- the nodes that are connected by the green links consist of a surface patch;
- a red link between two nodes means that the nodes are connected but not on the same surface patch.

Figure 4.21 (c) shows the reconstructed object from the AHR. The colors and textures of the table are also presented. A simple augmentation of the scene is to add some virtual object in the scene. As shown in Figure 4.21 (d), an arm chair, a table lamp and a piece of textured floor are integrated in the scene. Another important augmentation is the shape change of the objects. Figure 4.21 (e) gives the view with the table top enlarged. Finally, in Figure 4.21 (f), the texture of the table top has been changed to marble and the illumination effect of the table lamp on all objects is simulated.

3D Indoor Scene of the PAMI Laboratory

As with the table, the same experiment scheme is also applied to the 3D reconstruction result of the laboratory scene. Figure 4.22 (a) shows the triangular mesh generated from the face model given in Figure 4.20. The corresponding AHR constructed from the mesh is illustrated in Figure 4.22 (b). It is noticed that in the AHR, the green nodes connected each other with the green links to to form the floor plane. It is considered as the "navigable" area in the scene which would be used for path planning (refer to Section 5.5.2). Figure 4.22 (c) shows the reconstructed scene with textures and colors inverted from the original stereo images.

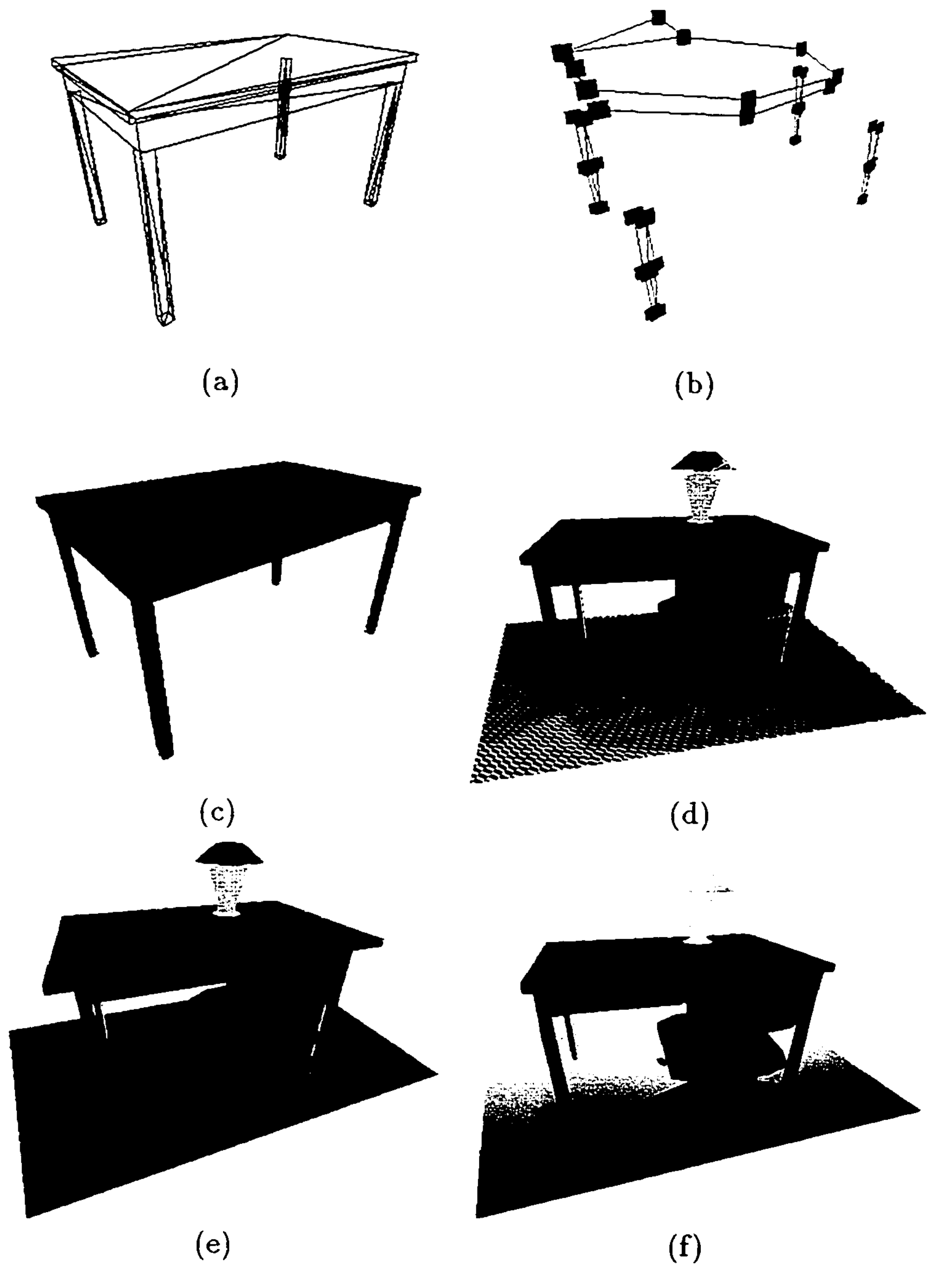


Figure 4.21: Experiments on the augmented reality of a laboratory table: (a) the triangular mesh model of the table; (b) the AHR of the table constructed from the triangular mesh; (c) the reconstructed geometry with colors and textures of the table; (d) the augmented scene with textured floor, a table lamp and an arm chair; (e) enlarged table top; (f) the table with marble top and virtual lighting effects.

Figures 4.22 (d) through (f) demonstrate three augmented views of the laboratory scene. The augmentations in the first scene include: (1) the incomplete table is replaced by the full table model that is used for camera pose determination; (2) the floor plane has been expanded outside of the door; and (3) the floor texture is changed. In the second picture, in addition to the texture change of the floor, a tetrahedron, a robot and a door are placed in the scene. The last augmented scene is built from the AHR in Figure 4.22 (b) but with only the geometrical properties preserved. The surface colors of all hypergraph nodes are arbitrarily assigned, and two virtual lighting sources are posed. Different from the previous view, the door is placed in its open position.

4.6.3 Error Analysis

There are a number of potential sources of error in the augmented reality construction, some of which are more significant than others. This section identifies the key contributors of the errors and discusses possible solutions for each of them.

Camera Distortion

It has been stated in Section 4.3.2 and Section 4.4 that a pin-hole camera model is assumed for pose calculation and model synthesis. For the thin camera lens, this may be applicable in the central area of the images. Nevertheless, a normal CCD camera with focal length of $6mm$-$8mm$ can have distortions up to 10 pixels around the perimeter. An example of such effect may be seen in Figure 4.23 which is the distorted image of a square grid. As can be seen, the lines in the center of the image are straight, whereas those at the edges are curved, especially when compared against the border. Due to such distortions, feature corners detected in

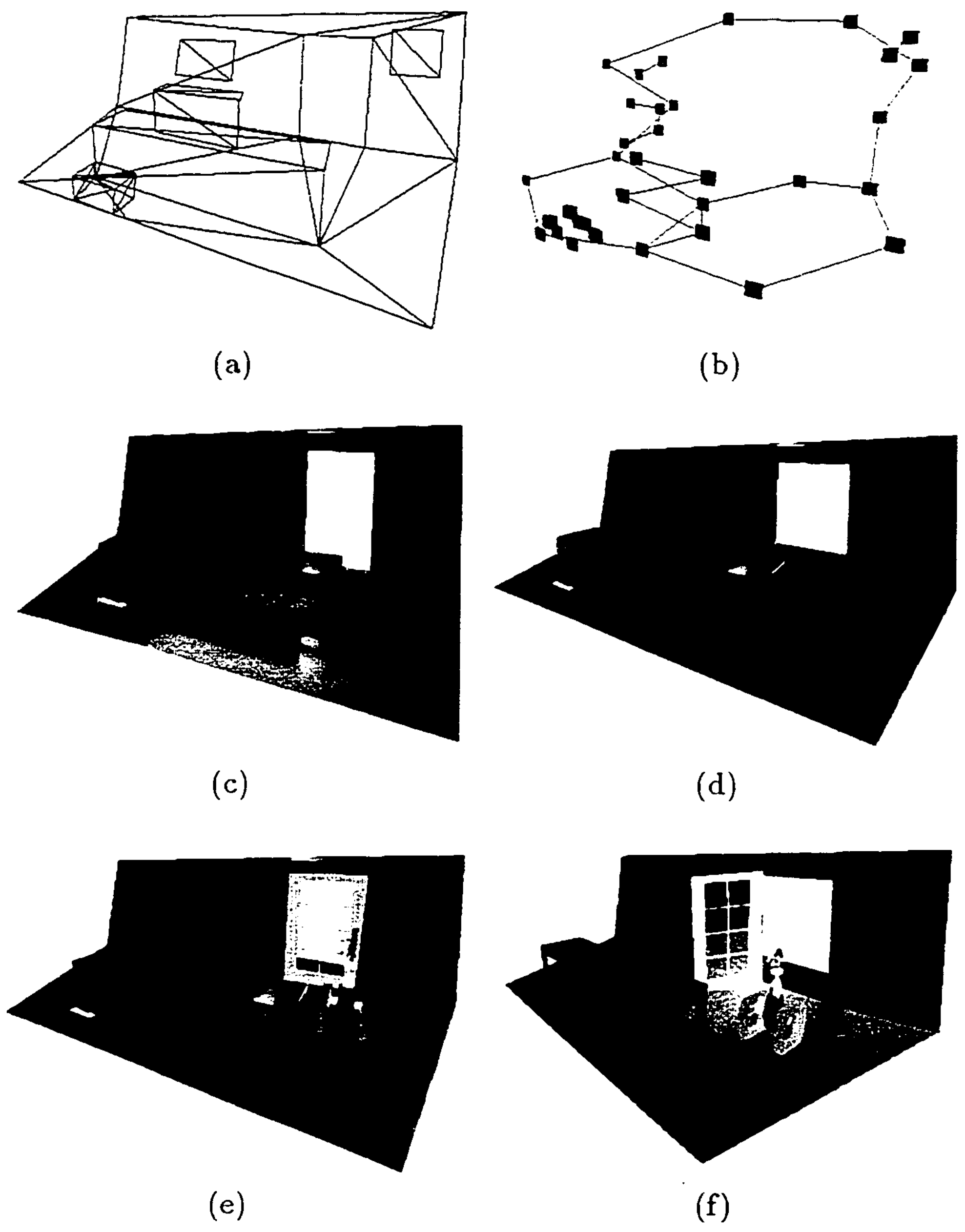


Figure 4.22: Experiments on the augmented reality of a laboratory scene: (a) the triangular mesh model of the laboratory scene; (b) the AHR of the scene constructed from the triangular mesh; (c) the reconstructed scene with colors and textures from the original images; (d) the augmented scene with a full table model and a new floor texture; (e) the scene with marble floor, a robot, a door and a tetrahedron; (f) simulated virtual texture and lighting effects.

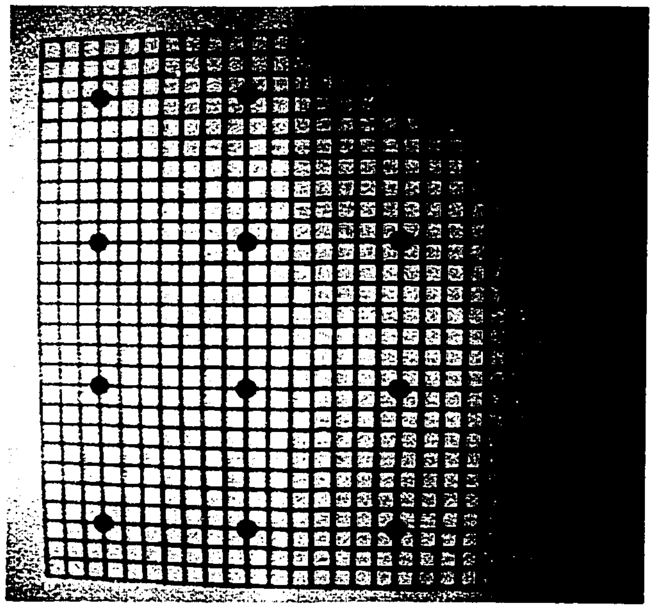


Figure 4.23: An image of a square grid taken by a camera with lens distortion.

the image may be located a few pixels away from their theoretical locations.

For pose determination, a simple solution is to put the calibration pattern around the central area of the image. This is probably the easiest solution, although not necessarily the desirable one since some accuracy will be still lost due the limited size of the pattern. Moreover, in 3D scene reconstruction, objects to be modeled may not be in the central locations. Alternatively, some forms of lens distortion calibration algorithm can be implemented. For example, Tsai's algorithm [91] evaluates the radial distortion by a parameter κ. If the origin of the image plane is located at the intersection with the optical axis, the correction can be expressed mathematically as:

$$x_U = x_D(1 + \kappa r^2), \qquad y_U = y_D(1 + \kappa r^2) \tag{4.25}$$

where $r = \sqrt{x_D^2 + y_D^2}$, (x_D, y_D) is the observed (i.e., distorted) point coordinate and (x_U, y_U) is the corrected coordinate. Note that the magnitude of the distortion depends upon the distance of the point from the center of the image.

Feature Extraction Error

The feature extraction procedure is a potential source of many errors, and even more, it can be the cause of errors found in other stages of AR construction. For example, if the edges are not properly extracted from the image data, the edge and corner features will be incorrect. Thus, the pose calculation and the vertex triangulation will be affected. The edge detection method adopted from [41] can effectively extract edge traces from digital images and achieve pixel level accuracy. However, most image digitizers are sized at 640×480 or 512×480 where even one pixel shift will result in significant error in the final result.

In a typical experiment of stereo triangulation, the object to be modeled is placed $2.5m$ away from a pair of CCD cameras (both with $16mm$ focal length) that are separated from each other by $600mm$. The digitizer's size is 512×480. The experiment shows that one pixel of shift of a 2D feature point will result in up to $5mm$ error in the range data obtained. In many applications, such a error level is considered fatal.

A number of sub-pixel image processing techniques are available in the literature to tackle the problem. By studying the behaviors of the lens and the digitizers which blur the edges in the images, many approaches are proposed. In [22], the edge detector utilizes operator impulse response to find the optimal zero crossing, which can reach sub-pixel accuracy. Another approach is to use a moment based edge operator [63], where the precision is achieved by correcting many deterministic errors (normally caused by non-ideal edge profiles) using a preset look-up-table.

Experiments have shown that edges can be located in digitized images to a twentieth of a pixel. The local energy approach is also very popular [67]. The philosophy of such approach is that edge or corner features always occur at points of maximum phase congruency. By the image itself and its Hilbert transform, a local energy function can be defined and the local maximum of the energy function occurs at points of maximum phase congruency.

Pose Calculation Error

The precision of the cameras' physical locations and orientations is also very important to ensure accurate model reconstruction. The approach discussed in Section 4.3.2 adopts the Fischler and Bolles algorithm [37] with a pose averaging mechanism and back-projection verification. It is robust and precise in most cases. However, it has been shown in [72] that the pose accuracy will decrease in the following situations: (1) when the distance between the calibration pattern and the camera increases; (2) when the tilt angle with respect to the camera increase; (3) when the incoming data set is perturbed.

An easy solution is to avoid the above situations when taking the pictures. This may be effective for experiments in laboratory environments but not quite practical in real applications. Alternatively, by increasing the number of reference points on the calibration pattern (i.e., increasing the correct candidates for least square optimization), we can effectively reduce the error, however, at higher computational cost.

The sensitivity of the algorithm stems from the fact that the accurate root of a quartic function is arbitrarily required. In fact, this quartic equation is a perturbed form of $(x-1)^4 = 0$. It is well known that this equation is very sensitive to perturbation and there is no stable closed form algorithm available until now. In [59], a Newton iteration method is proposed to solve a set of non-linear equations instead of finding the roots of the quartic polynomial. The method is proved to be stable in the sense that the final computational error is bounded by the initial error.

Calculation Error in Vertex Triangulation

The range information of the features is obtained by vertex triangulation. As shown in Equation 4.14, the range point $P_w = (X_w, Y_w, Z_w)^T$ is calculated from the homogeneous matrices of the two cameras and the two perspective projections of P_w. Here Equation 4.14 is rewritten in the form of

$$H_{AB} \cdot P_w = D_{AB} \tag{4.26}$$

where

$$H_{AB} = \begin{pmatrix} u_a A_{20} - A_{00} & u_a A_{21} - A_{01} & u_a A_{22} - A_{02} \\ v_a A_{20} - A_{10} & v_a A_{21} - A_{11} & v_a A_{22} - A_{12} \\ u_b B_{20} - B_{00} & u_b B_{21} - B_{01} & u_b B_{22} - B_{02} \\ v_b B_{20} - B_{10} & v_b B_{21} - B_{11} & v_b B_{22} - B_{12} \end{pmatrix}$$

$$D_{AB} = \begin{pmatrix} A_{03} - u_a A_{23} \\ A_{13} - v_a A_{23} \\ B_{03} - u_b B_{23} \\ B_{13} - v_b B_{23} \end{pmatrix}$$

It can be seen from Equation 4.26 that if H_{AB} is close to singular, it would be very difficult to accurately calculate P_w. Unfortunately, this sometimes happens in reality. For example, in a typical mobile robot application, two CCD cameras for stereo vision are mounted on the robot separated by less than $500mm$. For easy calibration and recognition. the two cameras are normally identical. Thus, the intrinsic camera parameters are the same. The scene to be sensed is usually 3 to 5 meters away from the robot. In the pair of stereo images, the same 3D point is viewed at similar orientations with similar distances away. With identical cameras, this means that the two homogeneous matrices H_A and H_B are very close. Moreover, the two projections $(u_a.v_a)$ and (u_b, v_b) are also very close. Therefore, we will have a nearly singular H_{AB}.

To avoid the problem, one can use a single camera mounted on a mobile robot and capture the stereo images at significantly different orientations by rotating and moving the robot.

Augmentation Error

Augmentation error occurs when integrating the virtual objects with the ones constructed from the real world. In most current AR systems, this error has been identified as one of the most significant problems that limit AR's applications. The objects in the augmented world must be properly aligned with respect to each other, or the illusion that the real and virtual worlds coexist will be compromised. More seriously, most applications *demand* accurate augmentation.

There are two types of augmentation error. The first is due to the incomplete information from the computer vision stage. Objects and/or textures that are partially or fully occluded in the given scenes have to be extrapolated. They may not correspond to the true information. In the constructed AR, typically when the combined world is viewed from an orientation that is much different from those of the available views, the errors may become significant.

The other type of the augmentation error is a propagation of the errors introduced in other stages. Theoretically, if the incoming virtual and real worlds are errorless, it would be rather straightforward to integrate them accurately. However, small errors from feature extraction, camera calibration or range calculation, which are even not observable in those phases, may be propagated in the augmentation stage. The errors produced in those processing stages can only be detected and compensated with the available data set. Nevertheless, some errors may not be noticeable in any of the available views. In an AR world, the viewer looks where he/she wants, and the system must respond with limited time delay. Therefore, errors beyond the estimation of the system may occur.

So far, augmentation errors are difficult to adequately control because of the critical requirements and numerous error sources. A practical work-around is to construct AR with many 2D views, which may reduce the possibility of error by using redundant information.

Chapter 5

AHR Applications in Computer Animation

As stated in Chapter 1, one of the objectives of the thesis research is to develop an efficient 3D object modeling algorithm that can automatically generate and control an animation. Given a target object represented as an attributed hypergraph, various forms of manipulations, such as morphing, animation construction and path planning, are expected to be performed by operations on the AHR's in a symbolic way. At the representation level, the equivalent task of finding the continuous shape change between two shapes becomes a search for the best matching between the two corresponding AH's, and then to extract the motion parameters that animate the source AH to the target one. In AHR, these motion parameters are interpreted in the form of a series of AH operators, which can be analyzed and re-applied to the AHR. In computer animation applications, with the above approach, vast labors associated with the human assisted generation of animation frames could be replaced by computer computation. The proposed system is given by Figure 5.1.

As illustrated in the figure, the AHR based animation system compares the source and the target AH's by an optimal subgraph isomorphism algorithm. Then based on the subgraph isomorphism result, the motions that would transform the first AH to the second one are partitioned into a global rigid part and a local non-rigid part. The two parts of the motions are interpreted into separate sets of AH operators, which are applied on the source AH to synthesize a continuous motion. The following sections describe the implemented system.

5.1 Optimal Subgraph Isomorphism for AHR

Given two AH's representing two states of an object under motion, before extracting the motion parameters, the correspondences between the entities of the two AH's (i.e., a *matching* between the two AH's) must be first established. Hence, we need to measure a certain *distance*, or discrepancy associated with the matching, such that a metric of AH's can be defined.

In the graph representation of vision applications, given two graphs (or hypergraphs), the "distance" between them is normally calculated by the following steps:

1. search the "best match" between them;
2. sum up the differences between each matched pair of graph entities;
3. measure (or assign) a penalty distance for unmatched nodes;
4. combine the results from 2. and 3. to obtain the total distance.

However, finding the "best match" between two graphs is well known to be an NP-hard problem in general. Adopted from [100], an optimal common subgraph isomorphism algorithm is used to reduce the average search space to a polynomial one. In the following, we begin with the essential notations used in graph matching.

5.1.1 Notations and Definitions

The definition of *isomorphism* for objects in category theory has been given in Definition 3.2. It is certainly applicable to graphs or hypergraphs which are objects in the AH category. Here a simplified definition of *isomorphism* for graphs and hypergraphs is provided.

Definition 5.1 The *isomorphism* problem for graphs (or hypergraphs) is to obtain a one-to-one mapping $u = (p, q)$ between two graphs (or two hypergraphs) $G_1 = (X_1, Y_1)$ and $G_2 = (X_2, Y_2)$, where X_1 and X_2 are the vertex sets, Y_1 and Y_2 are the edge sets (or hyperedge sets), such that $X_1 \overset{p}{\longleftrightarrow} X_2$ (written as $X2 = p(X_1)$) and $Y_1 \overset{q}{\longleftrightarrow} Y_2$ ($Y_2 = q(Y_1)$) are both one-to-one mappings. With u, G_1 is said to be *isomorphic* to G_2, denoted as $G_1 \sim G_2$.

Isomorphism preserves incidence, which means that if $G_1 \sim G_2$, for two vertices $v \in X_1$ and $v' \in X_1$, if $v \in H$ and $v' \in H$ where H is an edge (or hyperedge) of G_1, then $p(v) \in q(H)$ and $p(v') \in q(H)$.

Definition 5.2 Given two graphs (or two hypergraphs) G_1 and G_2, the *subgraph isomorphism* problem is to obtain a subgraph (or sub-hypergraph) g_1 from G_1 ($g_1 \subseteq G_1$) and a subgraph (or sub-hypergraph) g_2 from G_2 ($g_2 \subseteq G_2$) such that $g_1 \sim g_2$. g_1 or g_2 is called the common subgraph (or common sub-hypergraph) of G_1 and G_2.

Suppose that the common sub-hypergraph of hypergraphs G_1 and G_2, denoted as g_c ($g_c \subseteq G_1$ and $g_c \subseteq G_2$) is defined by the mapping $u = (p, q)$, then p (or q) will map a vertex (or a hyperedge) in G_1 to either a vertex (or a hyperedge) in G_2, or to a null vertex (or a null hyperedge). Those vertices and hyperedges mapped to null can be expressed by one or more hypergraph operators which could delete or generate vertices (e.g., *merge* and *dichotomy*) and hyperedges (e.g., *join* and *subdivision*).

Apparently, given G_1 and G_2, there may exist more than one pair of sub-hypergraphs which satisfy Definition 5.2. In most applications, we are interested in the "optimal" one among all pairs.

Definition 5.3 The *optimal subgraph isomorphism* problem for two graphs (or two hypergraphs) is to determine a common subgraph (or a common sub-hypergraph) g_c from G_1 and G_2 which achieves the minimum mapping cost associated with the common graph (or common sub-hypergraph). The mapping cost is subject to the criteria given in the task domain.

With normal hypergraphs, the common sub-hypergraph of G_1 and G_2 with the largest number of vertices is the optimal solution. For example, in Figure 5.2, the optimal sub-hypergraph of G_1 and G_2 is hypergraph g_c. However, for attributed graphs or attributed hypergraphs, the cost of mapping is subject to the definition of the attributes and the definition of the discrepancy among attributes.

As described in Section 3.3.1, attribute hypergraphs are implemented in a net-like data structure and are called DAH's (refer to Figure 3.3), which has the following properties:

- the data structure is layered;

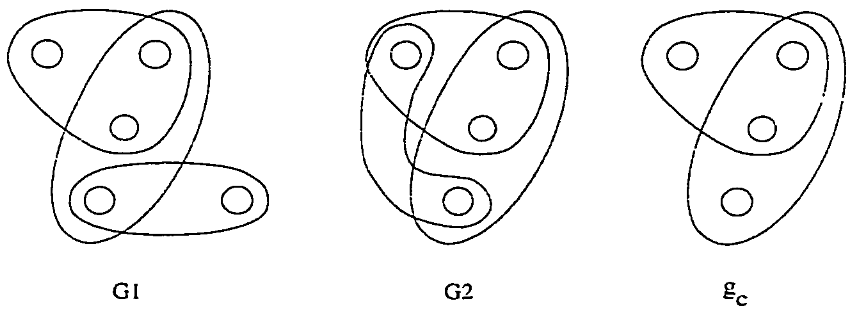


Figure 5.2: An example of the optimal subgraph of two hypergraphs.

- all the nodes on the same layer with the links among them consist of an attributed graph;
- in a DAH, the attributed graphs on different layers represent the same object at different resolution levels;
- in two DAH's of the same 3D object under motion, the attributes for the nodes from the same layer have the same type of geometrical, kinematic, physical or behavioral meanings.

Therefore, the layers in a DAH are separable, and in two DAH's representing the same 3D pattern, nodes on the same layer number are "comparable". Thus, the procedure to search for the optimal subgraph isomorphism for DAH's can be accomplished by subdividing the problem of matching two AH's into matching pairs of attributed graphs on different layers.

5.1.2 Cost of DAH Morphisms

Given two DAH's G_1 and G_2 with a morphism u to map G_1 to G_2, we shall define the mapping cost between them in this section based on the assumptions that:

- G_1 and G_2 are constructed using the triangular mesh method;
- G_1 and G_2 have the same number of layers;
- the attributes associated with the nodes on the same layer in either G_1 or G_2 have the same type(s).

Suppose that the number of layers in G_1 and G_2 is L. At layer l $(1 \leq l \leq L)$, the nodes in G_1 form an attributed graph denoted as G_{1l}, and the nodes in G_2 form an attributed graph denoted as G_{2l}. G_{11} and G_{21} correspond to the representative attributed graphs of the two triangular meshes respectively. Let the orders of G_{1l} and G_{2l} be m and n respectively. Without loss of generality, we assume that $m \geq n$. The cost of the mapping from G_{1l} to G_{2l}, denoted as c_l, can be written as:

$$c_l = \sum_m c(\Gamma_{G_{1l}}, u(\Gamma_{G_{1l}})) \tag{5.1}$$

where $\Gamma_{G_{1l}}$ is a DAH node in G_{1l}, $u()$ denotes a mapping and $u(\Gamma_{G_{1l}})$ is the mapping result of $\Gamma_{G_{1l}}$.

For each pair of nodes $\Gamma_{G_{1l}}$ and $u(\Gamma_{G_{1l}})$, if $u(\Gamma_{G_{1l}}) \in G_{2l}$, (which means that under the mapping $u()$, $\Gamma_{G_{1l}}$ of G_{1l} and $u(\Gamma_{G_{1l}})$ of G_{2l} are a matched pair), we have:

$$c(\Gamma_{G_{1l}}, u(\Gamma_{G_{1l}})) = \| A(\Gamma_{G_{1l}}), \quad A(u(\Gamma_{G_{1l}})) \| \tag{5.2}$$

where $A(\Gamma)$ denotes the attribute value of node Γ, $\| A, B \|$ denotes the normalized difference in values of A and B. For geometrical, kinematic and physical representations, typically, A and B are in the form of $(x_A, y_A, z_A, a_{1_A}, a_{2_A}, ...a_{n_A})$ and $(x_B, y_B, z_B, a_{1_B}, a_{2_B}, ...a_{n_B})$, where (x, y, z) are the 3D coordinates, and a_1 to a_n are other attributes such as curvature, orientation, mass density, damping factor, elastic coefficient, etc., then:

$$\| A, B \| = \sqrt{w_p(x_A - x_B)^2 + w_p(y_A - y_B)^2 + w_p(z_A - z_B)^2 + \sum_{i=1}^{n} w_i(a_{i_A} - a_{i_B})^2} \tag{5.3}$$

where w_p, w_i are the preset weights for the attributes.

If the topology of adjacency (for vertices) or incidence (for hyperedges) of $\Gamma_{G_{1l}}$ and $u(\Gamma_{G_{1l}})$ do not match, we consider that matching fails (i.e., it maps to a null node). In such a case, a penalty cost is assigned to the mapping:

$$c(\Gamma_{G_{1l}}, u(\Gamma_{G_{1l}})) = \lambda_l \tag{5.4}$$

where λ_l is the penalty value. Theorem 5.1 in the next section gives the criteria on choosing λ_l.

The cost of a morphism u between two DAH's G_1 and G_2 is then defined as the sum of the costs of mapping the attributed graphs on all layers:

$$c_u(G_1, G_2) = \sum_l w_l \sum_m c(\Gamma_{G_{1l}}, u(\Gamma_{G_{1l}})) \tag{5.5}$$

where w_l is the weighting factor for nodes on layer number l.

5.1.3 DAH Distance and Optimal Subgraph Isomorphism for DAH

Definition 5.4 The *optimal subgraph isomorphism* for attributed hypergraphs G_1 and G_2 is a mapping u from G_1 to G_2, such that: (1) the number of the matched nodes is maximized; (2) if (1) is satisfied, the cost $c_u(G_1, G_2)$ is minimized.

Definition 5.5 Associated with the optimal subgraph isomorphism T, the cost of morphism $c_T(G_1, G_2)$ from G_1 to G_2 is the distance between the G_1 and G_2:

$$d(G_1, G_2) = \min_u\{c_u(G_1, G_2)\} = c_T(G_1, G_2) \tag{5.6}$$

For layered DAH's, hyperedges on different layers can not be matched. A matching u between DAH's G_1 and G_2 with L layers can be obtained in the following way:

1. set $l = 1$;
2. match G_{1l} and G_{2l}, the result mapping being denoted as u_l;
3. $l = l + 1$, if $l > L$ go to step 4, otherwise go to step 2;
4. output $u = \{u_i | 1 \leq l \leq L\}$

Therefore, the distance between G_1 and G_2 can be defined as:

Definition 5.6 For DAH's G_1 and G_2 with L layers,

$$d(G_1, G_2) = \sum_{l=1}^{L} w_l\{\min_u \ c_l(G_{1l}, u(G_{1l}))\} \tag{5.7}$$

where $c_l(G_{1l}. T(G_{1l}))$ is the cost to map G_{1l} to G_{2l} under morphism u.

The distance can be also written in the following form:

$$d(G_1. G_2) = \sum_{l=1}^{L} w_l d_l(G_{1l}, T(G_{1l})) \tag{5.8}$$

where $d_l(G_{1l}. T(G_{1l}))$ stands for the distance between G_{1l} and $T(G_{1l})$. The search for an optimal subgraph isomorphism between two DAH's then becomes the search for optimal subgraph isomorphisms between L pairs of attributed graphs on different layers of the two DAH's.

Theorem 5.1 In Equation 5.4, for G_{1l} and G_{2l}, if

$$\lambda_l = k\xi = k \max_{\forall\Gamma_{G_{1l}}, \forall\Gamma_{G_{2l}}} c(\Gamma_{G_{1l}}, \Gamma_{G_{2l}}) \tag{5.9}$$

where $k \geq L$ and L is the number of nodes in G_{1l}, then a morphism u from G_{1l} to G_{2l} associated with the least mapping cost must have the maximum number of matched nodes.

[Proof]

Proof is made by contradiction. Suppose that for G_{1l} and G_{2l}, morphism u has the least mapping cost C_u with M matched nodes, morphism v has mapping cost C_v ($C_v > C_u$) with N matched nodes. For proving by contradiction, we assume that $M < N$. Since $C_u < C_v$:

$$\Rightarrow \sum_M c_u(\Gamma_{G_{1l}}, u(\Gamma_{G_{1l}})) + k(L-M)\xi < \sum_N c_v(\Gamma_{G_{1l}}, v(\Gamma_{G_{1l}})) + k(L-N)\xi \tag{5.10}$$

$$\Rightarrow \sum_N c_v(\Gamma_{G_{1l}}, v(\Gamma_{G_{1l}})) - \sum_M c_u(\Gamma_{G_{1l}}, u(\Gamma_{G_{1l}})) > k\xi(N-M) \tag{5.11}$$

$N > M$ and ξ is the maximum mapping cost between each pair of nodes, thus:

$$\Rightarrow \sum_N c_v(\Gamma_{G_{1l}}, v(\Gamma_{G_{1l}})) - \sum_M c_u(\Gamma_{G_{1l}}, u(\Gamma_{G_{1l}})) < \xi N \tag{5.12}$$

$$\Rightarrow k\xi(N-M) < \xi N \tag{5.13}$$

$$\Rightarrow k < \frac{N}{N-M} \leq N \leq L \tag{5.14}$$

which is contradicted to the assumption of $k \geq L$. Therefore, we must have $N \leq M$.

□

Theorem 5.2 For DAH's G_1 and G_2 with L layers, if u is the optimal subgraph isomorphism and g is the optimal common subgraph represented in DAH, then g_l is the optimal common subgraph of G_{1l} and G_{2l}, where g_l, G_{1l} and G_{2l} are the attributed graphs on layer l of g, G_1 and G_2 respectively $(1 \leq l \leq L)$.

[Proof]
Suppose that $u = \{u_l | 1 \leq l \leq L\}$ where u_l maps G_{1l} to G_{2l}. By definition (refer to Definition 5.5) we have:

$$d(G_1, G_2) = \sum_{l=1}^{L} w_l \{\min_u \ c_l(G_{1l}, u_l(G_{1l}))\}$$

thus, the cost of mapping on each layer must be minimized. Therefore, as associated with the distance, u_l has to be the optimal subgraph isomorphism between G_{1l} and G_{2l}.
□

5.1.4 Optimal Subgraph Isomorphism Algorithm for Attributed Graphs

Theorem 5.2 suggests that the optimal subgraph isomorphism of two DAH's can be obtained by calculating the optimal subgraph isomorphisms between pairs of attributed graphs on each layer of the DAH's.

The search for an optimal common subgraph isomorphism between two attributed graphs can be determined by a search strategy that employs a heuristic evaluation function [100]. It estimates the cost of each candidate solution for subgraph isomorphisms in the solution tree and finds the optimal one among of them.

Let N stand for the current node in the search tree ("node" here, which is broadly used in search tree related topics in artificial intelligence, is a different term from the "node" used in DAH). The heuristic function is made up of two components:

$$f(N) = g(N) + h(N) \tag{5.15}$$

where $g(N)$ measures the actual cost from the root node in the search tree to the current node N, and $h(N)$ is a heuristic estimation of the cost from N to a leaf node which is the descendant of the current node.

A solution is *optimal* if it has the least cost among all possible solutions. A search algorithm is *admissible* if, for any input graph, it always terminates in the optimal solution path whenever a path from the root to the goal exists. Suppose that there exists a heuristic function:

$$f^*(N) = g^*(N) + h^*(N) \tag{5.16}$$

where $g^*(N)$ is the real cost along the shortest path from the root to the node N and $h^*(N)$ always gives the exact cost along the shortest path from N to the goal, $f^*(N)$ is apparently *admissible.*

Unfortunately, such $f^*(N)$ does not exist explicitly for most real problems. We would like to find an f which is close enough to $f^*(N)$. In the attributed graph matching problem, it is easy to find a $g(N)$ equal to $g^*(N)$ while $h^*(N)$ is often impossible to compute. However, it is practical to determine whether or not $h(N)$ is bounded by the actual cost of $h^*(N)$, such that $\forall N$, $h(N) \leq h^*(N)$. With this kind of $f(N)$ and $g(N)$, we have an A^* algorithm which is well known to be admissible.

Let G_{1l} and G_{2l} be the attributed graphs on layer l of two DAH's G_1 and G_2, also let $g_1 = \{v_{1k} | 1 \leq k < i\}$ and $g_2 = \{p(v_{1k}) | 1 \leq k < i\}$ be the vertex sets of the matched subgraphs of G_{1l} and G_{2l} respectively. We can form a solution tree such that the matched vertex pairs at i^{th} searching level can be represented by a node $N = \{(v_{1k}, p(v_{1k})), 1 \leq k < i\}$ where p is a one-to-one mapping between g_1 and g_2. In N, each element is a tuple (v_{1i}, v_{2j}) where vertex v_{1i} in G_{1l} is matched to vertex v_{2j} in G_{2l}. Then $g(N)$ can be written as:

$$g(N) = \alpha \cdot \sum_{v_{1k} \in g_1} d[v_{1k}, p(v_{1k})] + \beta \cdot \sum_{v_{1k}, v_{1j} \in g_1, k \neq j} d(E[v_{1k}, v_{1j}], E[p(v_{1k}), p(v_{1j})]) \quad (5.17)$$

where $E[v_{1k}, v_{1j}]$ denotes the edge between vertex v_{1k} to v_{1j}. If for $v \in g_1$, $p(v) = \phi$, then:

1. $d[v, p(v)] = \lambda_l$, and
2. $d(E[v, v'], E[p(v), p(v')]) = \lambda_l$

and if for $E[v_{1k}, v_{1j}] \in g_1$, $E[p(v_{1k}), p(v_{1j})] = \phi$, then $d(E[v, v'], E[p(v), p(v')]) = \lambda_l$

$h(N)$ of Equation 5.15 is adopted from [100] which consists of two parts:

$$h(N) = a(N) + b(N) \quad (5.18)$$

where

$$a(N) = \sum_{v_{1p} \in M_1} \min_{v_{2q} \in M_2} \{d(v_{1p}, v_{2q}) + \sum_{v_{1k} \in g_1} d(E[v_{1p}, v_{1k}], E[v_{2q}, p(v_{1k})])\} \quad (5.19)$$

$$b(N) = \sum_{v_{1s}, v_{1t} \in M_1, s<t} \min_{v_{2q}, v_{2r} \in M_2, q \neq r} d(E[v_{1s}, v_{1t}], E[v_{2q}, v_{2r}]) \quad (5.20)$$

in which:

M_1 is the set of unmatched vertices of G_{1l}, i.e., $M_1 = X_{1l} \backslash g_1$ if $G_{1l} = (X_{1l}, Y_{1l})$.

M_2 is the set of unmatched vertices of G_{2l}, i.e., $M_2 = X_{2l} \backslash g_2$ if $G_{2l} = (X_{2l}, Y_{2l})$.

v_{1p}, v_{1k}, v_{1s}, v_{1t}, v_{2q}, v_{2r} represent vertices and $E[v_1, v_2]$ denotes the edge between v_1 and v_2.

In Equation 5.18, $a(N)$ favors a mapping with the minimal cost between the unmatched vertices of G_{1l} and the unmatched vertices of G_{2l}; $b(N)$ favors the optimal mapping between the unmatched edges of G_{1l} and the unmatched edges of G_{2l}.

5.2 Attributed Hypergraph Based 3D Motion Analysis

AHR provides a practical solution for the motion analysis of a moving object. We assume the input data are the 3D shapes or range information of an object represented by attributed hypergraphs. Given two attributed hypergraphs which represent two states of an object, the common parts or the parts with attribute changes can be found using the optimal subgraph isomorphism algorithm described in the last section. In the common subgraph, node-to-node correspondences from the source hypergraph to the target hypergraph with attribute changes can be obtained. On the other hand, those unmatched nodes can be characterized by qualitative AH operators. With the functors defined in Section 3.3.4, the qualitative and the quantitative differences correspond to sequences of AH operators. By representing the motions with operators, process abstractions and other symbolic operations on AHR are possible.

In this section, free form motions are decomposed into two parts: the rigid part (kinematic transformations with 6 degree of freedoms) and the non-rigid part (deformations). We assume that non-rigid deformations are confined in local areas and are non-significant compared with rigid transformations. Therefore, the algorithm first estimates the global rotation matrix and translation vector that are applied to all or part of the vertices and the hyperedges, and then extracts the AH operators corresponding to the local deformations.

5.2.1 Kinematic Transformation Determination

With the result of optimal subgraph isomorphism, the correspondences of nodes between the two DAH's are known. Then the task of rigid motion determination becomes a traditional one. Supposing that $(p_1, p_2, ..., p_n)$ be the observed 3D points relative to the world coordinate system, and $(p_1', p_2', ..., p_n')$ be the corresponding 3D points observed in the same coordinate system with the object moved, we wish to compute a rotation matrix R and a translation vector T associated with this motion.

This problem has been well studied in both solid geometry and computer vision [3, 53, 61, 82]. Suppose that the coordinate of a 3D point in the first scene is $p_i = (x_i, y_i, z_i)$, and in the second scene is $p_i' = (x_i', y_i', z_i')$, then the desired R and T should give $p_i' = Rp_i + T$. Though R is a 3×3 matrix with 9 elements, it can be represented by three rotation angles since R has to be orthogonal. In [3], an algorithm that would compute the R and T which minimize $\sum_{i=1}^{n} ||p_i' - (Rp_i + T)||$ is given. When the non-rigid motions are insignificant compared with the rigid ones, this algorithm is quite robust. In case the algorithm fails, an alternative method is applied to obtain R by computing the three angles.

Following the convention in robotics, the rotation matrix R is represented by three angles. denoted as O, A and T:

$$R = \begin{pmatrix} \cos O \sin T - \sin O \sin A \cos T & \cos O \cos T + \sin O \sin A \sin T & \sin O \cos A \\ \sin O \sin T + \cos O \sin A \cos T & \sin O \cos T - \cos O \sin A \sin T & -\cos O \cos A \\ -\cos A \cos T & \cos A \sin T & -\sin A \end{pmatrix} \tag{5.21}$$

Theoretically. there are only 6 unknowns to be solved in R and T. If we can obtain at least four pairs of correspondencies, a linear approach can be applied to solve the equations with the least square method and to compute R and T directly. The more matched pairs we can find, the more accurate the approximation of the rigid part of the motion by the computed R and T.

The functions to solve R and T can be expressed as:

$$\begin{aligned} x_i' &= r_{11}x_i + r_{12}y_i + r_{13}z_i + d_x \\ y_i' &= r_{21}x_i + r_{22}y_i + r_{23}z_i + d_y \\ z_i' &= r_{31}x_i + r_{32}y_i + r_{33}z_i + d_z \end{aligned} \tag{5.22}$$

where $p_i = (x_i, y_i, z_i)$ and $p_i' = (x_i', y_i', z_i')$ are the attribute values of a pair of matched nodes. n is the total number of matched pairs, $1 \leq i \leq n$.

Due to the presence of non-rigid motions and noises/errors, the resulting R calculated by the linear least square method is unlikely orthogonal. A simple algorithm is applied to calculating the closest O, A and T from the raw R as shown below. followed by a reconstruction of an orthogonal R from the estimated O, A, T by Equation 5.21. The C++ source code to estimate O, A and T is:

```
void R_to_OAT(Matrix R(3, 3), double &O, double &A, double &T)
{ double x, y, cA;
```

```
A = asin( (double) (-R(3,3)) );  // A (radian)
cA = cos((double) A);
A *= DEG;

if ( fabs(cA) < TINY )
  { // in case that A ~= +/- 90 deg, there are infinite
    // numbers of solutions then assume T = 0
    x = R(1,2);
    y = R(2,2);
    if (fabs(x) > TINY)
      O = atan2(y,x)*DEG;
    else
      O = ((y > 0) ? 90 : -90);
    T = 0;
  }
else
  { x = -R(2,3)/cA;
    y = R(1,3)/cA;
    if (fabs(x) > TINY)
      O = atan2(y,x)*DEG;
    else
      O = ((y > 0) ? 90 : -90);

    y = R(3,2)/cA;
    x = -R(3,1)/cA;
    if (fabs(x) > TINY)
```

```
        T = atan2(y,x)*DEG;
      else
        T = ((y > 0) ? 90 : -90);
    }
  return;
}
```

5.2.2 Non-rigid Motion Interpretation

Non-rigid motion interpretation is performed based on the determined transformation for rigid motions (R and T). Each set of parameters computed for non-rigid motions is valid locally and confined to a node and its neighbors only. Non-rigid motions are classified into quantitative deformations and qualitative deformations.

Quantitative Deformation

Quantitative deformations correspond to the differences of attribute values between pairs of matched nodes in DAH. "Quantitative" means that each pair of nodes have the same connectivity property; the only differences are the attribute values. In other words, quantitative deformations result in no change in the AHR topology.

For each pair of matched nodes, from the R and T of rigid motions, an attribute displacement can be obtained. Suppose that in two DAH's, $p = (x, y, z)$ and $p' = (x', y', z')$ are the geometrical attributes of a pair of matched nodes (denoted as n and n'), R and T are the determined rigid motion parameters, we can have $p^* = (x^*, y^*, z^*)$ where $p^* = R \cdot p + T$. The quantitative deformation associated with node n is defined as an attribute transition operator $f_{a_n}()$, where:

$$f_{a_n}(p) = p + (x' - x^*, y' - y^*, z' - z^*)^T \tag{5.23}$$

Qualitative Deformation

The qualitative deformations occur when there are unmatched nodes between two DAH's. These nodes either merge with others or emerge as new ones. They are *qualitative* in that they change the hypergraph's topological structure.

The *merge* and *dichotomy* of vertices, and the *subdivision* and *join* of hyperedges, (refer to Section 3.3.3), are applied to manipulate the disappearing and emerging nodes in DAH. Given two DAH's G and G', our task is to extract the operator set Op which transforms G to G'. Op consists of one or a sequence of the primary operators.

For simplicity, the extraction of the operators is rule-based in a syntactic manner. A set of primary deformations on elementary nodes and elementary edges are defined corresponding to the four primary operators. Other qualitative deformations can be defined as their combinations.

- The *subdivision* on the edges of a triangle can be categorized into three types: one-edge, two-edge and three-edge subdivisions that subdivide one, two or three edges of the triangle respectively (refer to Figure 5.3 (a)).
- The *join* of two edges implies the deleton of the common vertex of the two edges. Any other edges connected to the vertex should be removed as well (see Figure 5.3 (b)). In a triangular mesh, two connected edges can relate to up to four triangles. The join of these two edges causes the re-arrangement of these triangles into two.

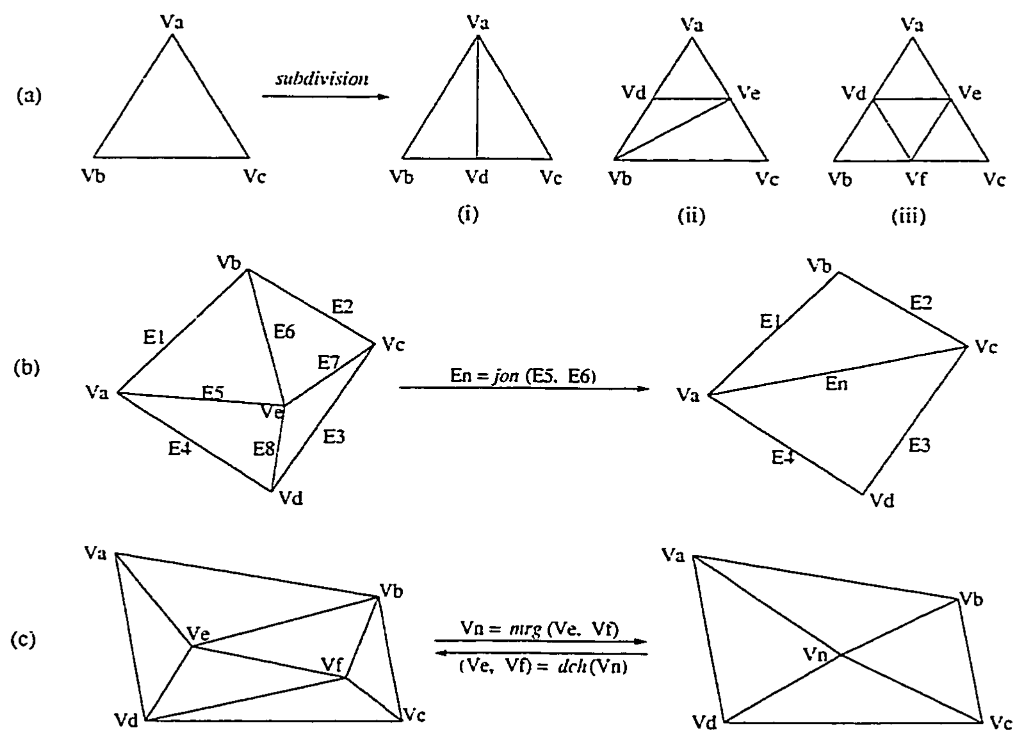


Figure 5.3: (a) three cases of edge *subdivision* on a triangle: (i) one-edge; (ii) two-edge; (iii) three-edge; (b) edge *join* on a triangular mesh; (c) vertex *dichotomy* and *merge* on a triangular mesh.

- The *dichotomy* operator brings two extra triangles in the mesh (refer to Figure 5.3 (c)).

- The *merge* of two vertices implies the deletion of the edge connecting two vertices. Up to two triangles that are incident to the edge will be deleted. An example is shown in Figure 5.3 (c).

Suppose that the two input DAH's are G_1 and G_2 and their common sub-hypergraph is g. The operator extraction is performed incrementally with the following steps:

1. traverse all the nodes in g and search for unmatched nodes that are adjacent

to at least one node in g;

2. if an unmatched node n_u is adjacent to node n_g in g while n_u is in G_2, then:
   (a) apply an operator op_0 which is either a *sub* or a *dch* to n_g in G_1 to yield two new nodes n_1 and n_2;
   (b) apply attribute transition operators op_1 and op_2 to n_1 and n_2 in G_1 respectively, such that $A(op_1(n_1)) = A(n_g)$ and $A(op_2(n_2)) = A(n_u)$, where $A(n)$ denotes the attribute value of node n;
   (c) add n_u to g and enlist op_0, op_1 and op_2 to the solution queue;
3. if an unmatched node n_u is adjacent to node n_g g and n_u is in G_1, then:
   (a) apply an operator op_0 which is either a *join* or a *merge* to n_g and n_u in G_1 to yield one new node n_p;
   (b) apply an attribute transition operator op_1 to n_p in G_1 such that $A(op_1(n_p)) = A(n_g)$;
   (c) enlist op_0. op_1 to the solution queue;
4. go back to step 1 with updated G_1, G_2 and g until there is no unmatched node left;
5. optimize the solution list by eliminating operators that cancel each other;
6. output the solution list.

5.2.3 A Simple Example of AHR Based Motion Analysis

Figure 5.4 gives a simple case where non-rigid motion is involved in the AHR of a piece of simulated terrain shown in Figure 3.13. Figure 5.4 (a) is the triangular

Table 5.1: The matching result and the corresponding AH operators extracted in motion analysis for two simulated terrains.

Feature in scene #1	DAH #1	Feature in scene #2	DAH #2	AH operator
peak	v_a	*peak 1*	v_a	f_a
hill corner 1	v_b	*hill corner 1*	v_b	f_a
hill corner 2	v_c	*hill corner 2*	v_c	f_a
hill corner 3	v_d	*hill corner 3*	v_d	f_a
plain corner 1	v_e	*hill corner 4*	v_e	f_a
plain corner 2	v_f	*hill corner 5*	v_f	f_a
N/A	ϕ	*peak 2*	v_g	$\wedge E[v_d, v_f] \& f_a$
hill side 1	H_1	*hill side 1*	H_1	f_a
hill side 2	H_2	*hill side 2*	H_2	f_a
hill side 3	H_3	*hill side 3*	H_3	f_a
plain	H_4	*mound #2*	H_4	f_a

mesh approximation of a scene. Suppose that after an earthquake, the same area deforms into the shape as shown in Figure 5.4 (b). Figures 5.4 (c) and (d) show the AHR's and Figures 5.4 (e) and (f) are the DAH's of the two scenes respectively.

With optimal subgraph isomorphism performed on the two DAH's, 6 pairs of matching nodes on the bottom layer are found (refer to Table 5.1). For these 6 pairs of nodes, there are only quantitative changes. The rigid motion parameters R and T are easily computed. For the emerging vertex v_g, from its connectivity with other elementary nodes, it can be inferred by the pre-set rules that v_g comes from the *subdivision* of edge $E[v_d, v_f]$ in the source DAH. On the hyperedge layer, it is interesting that all nodes are matched successfully. This indicates that the qualitative deformation identified for the elementary graph is insignificant since it does not propagate to the hyperedge layer.

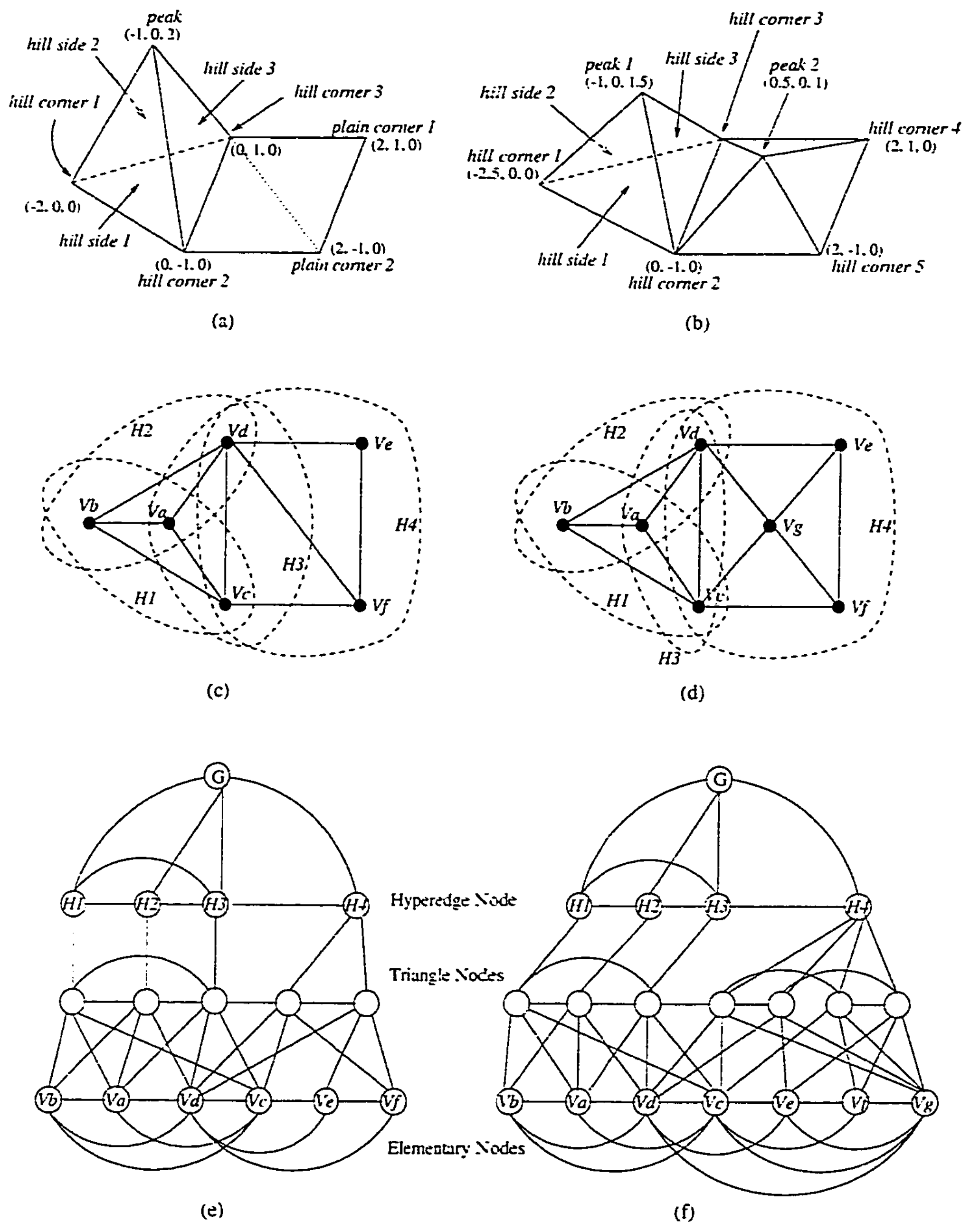


Figure 5.4: An example of motion analysis on the triangular meshes of a natural terrain: (a) scene #1: a hill and a plain; (b) scene #2: a hill and a small mound: (c) the AHR of scene #1; (d) the AHR of scene #2: (e) DAH #1: the representation of scene #1; (f) DAH #2: the representation of scene #2.

5.3 3D Morphing

One of the most important applications of AHR based motion analysis is automatic 3D morphing generation. Morphing means the metamorphosis from one state of an object (the source) to another state (the target) via continuous deformations.

AHR provides an ideal tool to construct 3D morphing sequence. As in the motion analysis, given two DAH's, a set of operators can be extracted. Moreover, for morphing, we also need to reconstruct a sequence of 3D objects from their DAH's with continuous motion. The AHR helps us to design an intelligent tool that only requires us to tell it "what" to do rather than "how" to do it.

5.3.1 Morphing Control

If two states of the same object are represented by attributed hypergraphs in the form of DAH's, the morphing corresponds to a sequence of transformations on the source DAH such that it finally changes itself into a DAH which is isomorphic to the target DAH. The problems in AHR based morphing to be investigated are:

1. how to determine a morphing path between two objects; and
2. how to justify the qualitative and the quantitative changes in the morphing.

Feature Preserving Morphing

To produce the metamorphosis of an object from one state to another one, there may exist many paths. There are usually certain constraints controlling the morphing. The *volume preserving* and the *surface preserving* constraints are the popular ones in applications. These methods usually use Minkowski sums or a merged topology to control the time-varying object. A DAH based morphing is different in that it uses *feature preserving* constraints. It is more flexible than *volume preserving* and the *surface preserving* methods in that the features to be preserved can be different in different applicaiton domains.

In AHR based morphing, given the initial DAH G_1 and the final DAH G_2, an optimal subgraph isomorphism algorithm first finds the optimal common sub-hypergraph of the two DAH's. Since the DAH distance measure applied in the optimal subgraph isomorphism algorithm is confined by the type of selected features, the morphing path generated follows the path with the minimal cost subject to the feature type. In other words, the algorithm attempts to find the path that minimizes the cost of matching associated with the selected feature type. Thus, it is feature preserving morphing.

The operator sequence that performs the transformations between the two DAH's can be determined just as we have illustrated in motion analysis. The operator extraction begins with the global motion parameters R and T from all the matched nodes. The least square method in the R and T estimations minimizes the total local displacement for the matched nodes. Since there are only quantitative changes for those matched ones, the qualitative information associated with them is well preserved. Meanwhile, the optimal subgraph isomorphism algorithm minimizes the number of unmatched nodes since they will introduce penalties that have the highest cost in matching. Hence, during the process of deforming G_1 into G_2 by applying the operator sequence, the qualitative features in the DAH are maximally preserved.

Frame Generation

The automatic morphing generation is based on the definition of distance between DAH's given in Section 5.1.3. If $d(G_1, G_2)$ is the distance between two DAH's G_1 and G_2 as defined in Equation 5.6, we can use linear interpolation to generate continuous morphing frames between G_1 and G_2. Suppose that in the morphing, the total time from pattern G_1 to G_2 is T. Given any time t such that $0 \leq t \leq T$, the corresponding morphed DAH at t, denoted as $G(t)$ can be expressed as:

$$G(t) = G_1 + [\frac{t}{T} \bullet (G_2 - G_1)] \tag{5.24}$$

where $G_2 - G_1$ signifies the difference between G_1 and G_2 measured by the AH operator set, and $\bullet$ means using its left-hand-side as the arithmetic operator to quantify its right-hand-side AH operator set.

The connotation of Equation 5.24 is that $G(t)$ can be built by applying a proper set of AH operators that varies with t. Suppose that the set of operators to transform G_1 to G_2 is $Op = \{op_1, op_2, ..., op_n\}$, where op_i $(1 \leq i \leq n)$ can be any one of the primary AH operators defined in Section 3.3.3, and the set of operators that transform G_1 to $G(t)$ is expressed as $Op(t) = \{op_1(t), op_2(t), ..., op_n(t)\}$. Referring to Equation 5.24, $op_i(t)$ is derived from op_i by:

- if op_i is an attribute transition operator, $op_i(t)$ is also an attribute transition operator:

$$op_i(t) = (1 - \frac{t}{T})op_i \tag{5.25}$$

- if op_i is *intersection*, *dichotomy* or *subdivision*, then:

$$op_i(t) = \begin{cases} op_i & \text{if } 0 < t \leq T \\ I_{op} & \text{otherwise} \end{cases} \tag{5.26}$$

  where I_{op} denotes the identity operator that does not change the input AH; if op_i is accompanied by an attribute transition operator f_{a-op_i}, then:

$$f_{a-op_i}(t) = (1 - \frac{t}{T})f_{a-op_i} \tag{5.27}$$

- if op_i is *union*, *merge* or *join*, then:

$$op_i(t) = \begin{cases} I_{op} & \text{if } 0 \leq t < T \\ op_i & \text{otherwise} \end{cases} \tag{5.28}$$

  where I_{op} denotes the identity operator that does not change anything; if op_i is accompanied by an attribute transition operator f_{a-op_i}, we have:

$$f_{a-op_i}(t) = (1 - \frac{t}{T})f_{a-op_i} \tag{5.29}$$

Examples of qualitative transition operator $op_i(t)$ are shown in Figure 5.5 (a) and (b).

Theorem 5.3 If $G(0) = G_1$ and $G(T) = G_2$, for $0 \leq t \leq T$,

$$d(G_1, G(t)) + d(G(t), G_2) = d(G_1, G_2) \tag{5.30}$$

Proof:

A. When $t = 0$:

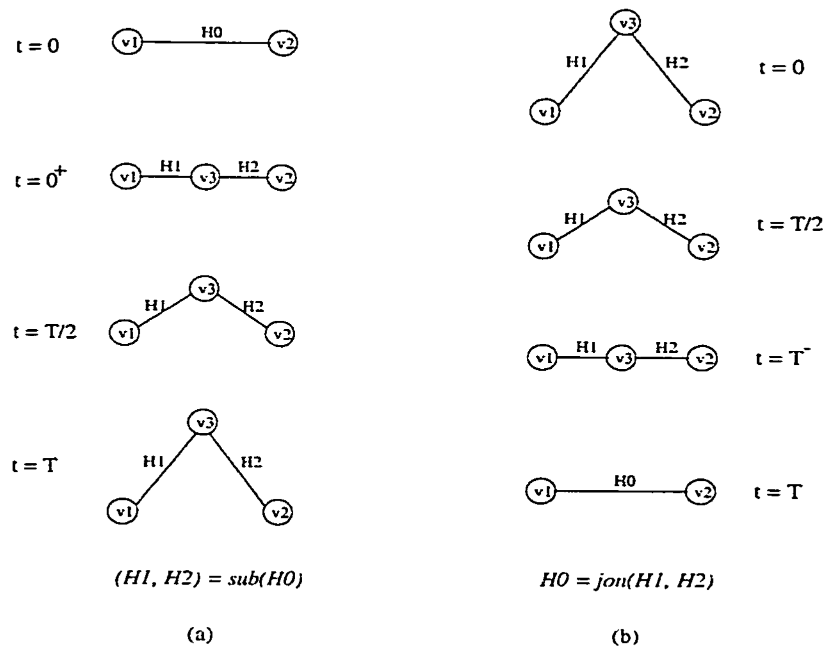


Figure 5.5: Examples of continuous morphing with qualitative transition operators on simple graphs: (a) *subdivision* and (b) *join*.

We have $d(G_1, G(t)) = 0$ and $d(G(t), G_2) = d(G_1, G_2)$, therefore $d(G_1, G(t)) + d(G(t), G_2) = d(G_1, G_2)$.

B. When $0 < t < T$:

Suppose that the optimal common sub-hypergraph of $G(t)$ $(0 < t < T)$ and G_2 is g, their number of layers is L, the number of nodes on layer l of g is N_l $(1 \leq l \leq L)$, between $G(t)$ and G_1 the total distance induced by unmatched nodes is λ_1, between $G(t)$ and G_2 the total distance induced by unmatched nodes is λ_2. we shall have:

$$d(G_1, G(t)) = \lambda_1 + \sum_{l=1}^{L} w_l \sum_{i=1}^{N_l} d(\Gamma_{i1}, \Gamma_i(t)) \tag{5.31}$$

$$d(G(t), G_2) = \lambda_2 + \sum_{l=1}^{L} w_l \sum_{i=1}^{N_l} d(\Gamma_i(t), \Gamma_{i2}) \tag{5.32}$$

$$d(G_1, G_2) = \lambda_1 + \{\sum_{l=1}^{L} w_l \sum_{i=1}^{N_l} d(\Gamma_{i1}, \Gamma_{i2})\} + \lambda_2 \tag{5.33}$$

where Γ_{i1}, $\Gamma_i(t)$ and Γ_{i2} are the i^{th} matched nodes in G_1, $G(t)$ and G_2 respectively. From Equation 5.3, it is obvious that if G_1, $G(t)$ and G_2 are the representations as prescribed in Chapter 3, we have:

$$d(\Gamma_{i1}, \Gamma_i(t)) + d(\Gamma_i(t), \Gamma_{i2}) = d(\Gamma_{i1}, \Gamma_{i2}) \tag{5.34}$$

From Equation 5.31, 5.32, 5.33 and 5.34, we can have Equation 5.30.

C. When $t = T$:
Since $d(G_1, G(t)) = d(G_1, G_2)$ and $d(G(t), G_2) = 0$, thus $d(G_1, G(t)) + d(G(t), G_2) = d(G_1, G_2)$.
□

5.4 Intelligent Animation

The computer graphics techniques allow the construction of 3D graphical objects from 3D models. In a 3D space, scenes are viewed/projected using virtual cameras and they may be lightened by virtual light sources. Based on the objects generated by computer graphics, computer animation techniques deal with the problems of how to express time dependence in the above scenes and how to make them evolve over time. In this section, animation based on AHR is addressed to show that machine intelligence algorithms can be integrated into the AHR framework to enhance computer animation.

5.4.1 Procedural Animation

In general, computer animation has the following three types of methodologies:

- In *naive animation*, the key features of animation in each of the frames are given to the computer. This means that the computer is used as a high level painting tool for drawing.
- The second and most used method in the literature is called *keyframe animation*. It consists mainly of a certain number of frames, called the *keyframes*, given to the computer by the animator. Then the computer derives the other frames using shape interpolation procedures.
- The third approach is an extension of the keyframe animation, called *procedural animation*. In this method, motion is described as a certain type of script language, mainly stating the starting status and the goal together with the constraints. The animator only provides limited amount of supervision, while the details of the motion are computed according to known procedural knowledge and the given constraints by the computer.

In the third type of animation, motion is derived from the task descriptions and the constraints. Typical constraints are the laws of physics. For example, to generate the animation corresponding to the motion of a falling particle with initial height h, the motion is governed by the equation $y = h - \frac{1}{2}gt^2$.

The procedural methodology requires the least user interaction but until now has few research results due to the difficulties in data representation at knowledge level. AHR provides a unified representation for both object shapes and knowledge. Therefore, it is ideal for procedural animation:

- the object to be animated is represented by attributed hypergraph;
- the constraints and laws of the motion are integrated in the attributed hypergraph as attributes and operators;
- the initial state and the final state of the animation are given in the form of two attributed hypergraphs;
- the animation procedure are calculated by matching the two given attributed hypergraphs with optimal subgraph isomorphism, and are represented by a sequence of AH operators applied to the initial AH.

The AHR framework is versatile because different types of animations can be adopted according to different application contexts. If geometrical and kinematic features are applied as the constraints to the AHR. and several key animation frames are given, keyframe animation can be accomplished by simple interpolations. In such cases, the details of the animation are fully controlled by the animator via keyframe editing. However, if such keyframes are not provided for reducing the animator's workload, or for generating animations that are purely controlled by natural laws, AHR can be re-organized to achieve procedural animation:

1. the attributes are chosen as sets of parameters that represent the object's physical states;
2. the constraints in animation are given in the form of a set of partial differential equations;
3. the initial state and the targeting state are known;
4. the physical states at any time can be computed numerically;

5. the motion frames are reconstructed based on the computed physical states.

The advantage of using algorithmic laws as the constraint of animation is that it will give the exact animation provided that the constraints follow the natural laws (e.g., the kinematics and/or dynamics laws). Theoretically, by interpreting algorithmic description, the animation frames are computed continuously, i.e., there is no need to use interpolation required by keyframe animation. Alternatively, in cases where the real time animation is required, interpolation can still take effects to reduce the computation cost.

The laws applied to animation are not rigid. The applicable laws can be dependent on the global state variables which are accessed by the defined procedure during the animation process. For most animation tasks this feature is very important for complex animation.

Unfortunately, there are several drawbacks to procedural animation:

- it could be very difficult to find the equations for an arbitrarily given motion that correspond to the complex natural laws;
- when several motions are joined, such as a linear accelerated motion followed by a circular motion, animation generated may be jerky at the transition points;
- exact computation of physically based motions is often impossible and some approximation methods such as the finite element approach are required;
- real-time computing is difficult due to the huge amount of computer resource required.

5.4.2 Knowledge from Automatic Process Abstraction

With a generic representation of the object/scene model, the advantage of AHR in processing on the knowledge level is obvious. Process learning can be integrated as an intelligent tool for procedural animation. The learning of an animation process is performed by abstracting primary AH operators into compound, high-level ones. This is equivalent to the process of extracting the qualitative information associated with a set of AH operators. For example, from a sequence of attributed hypergraphs which represent a walking kitten, the knowledge of how a kitten walks can be "learned" in the form of a set of AH operators and then applied to the AHR of a tiger.

The primary operators listed in Section 3.3.3 are derived from the sequence of attributed hypergraphs provided. Though the representation is on a symbolic level, the operator sequence is not the kind of "knowledge" that is ready for reuse (recall) since much of the high-level information is implicit or scattered among the low-level ones. Therefore, the extracted operator set is further abstracted by preserving only the operators applied on high level nodes and composing primary operators into compound ones. The general composition rules are:

- operators applied on the same hyperedge are integrated into a compound one;
- from the operator set *Op*, the qualitative operators are preserved as pieces of "knowledge";
- the quantitative changes are quantified into relative set membership such as $\{large, mediumlarge, medium, mediumsmall, small\}$.

For example, in the kitten-tiger example, a kitten's head, tail and limbs are represented by different hyperedges. During the kitten's walking, the relative movements of its head, tail and legs are analyzed. The motion pattern, in the form of a set of compound operators, can be easily applied to the AHR of another similar animal having the same number of head, tail and limbs.

5.4.3 AHR Based Path Planning

The unified representation of an object and its environments also makes it easy to integrate collision detection, path planning and navigation algorithms with the animation. With the AH description of the scene obtained from triangular meshes, obstacles and the peaks, ridges, etc., of the scene can be registered. In more sophisticated cases, the obstacles are not stationary. Their motion features are represented by attributes in AHR, acting as extra constraints to the search of a trajectory.

In AHR, a scene is represented by attributed hypergraph with a layered data structure (DAH). The nodes on the same layer of the DAH consist of an attributed graph. Different layers of the DAH represent the same scene at different resolution levels. Therefore, at a preferred resolution level, if the starting location and the target are known, path planning is equivalent to a typical problem in graph theory: to find the shortest *path* between two vertices in an attributed graph where the attribute values are the lengths. Apparently, an advantage of AHR based path planning is that it can be performed on various resolution levels.

Suppose that the attributed graph is $G = (V, E)$, the starting vertex is u_0 and the ending vertex is v_0, the length between two connected vertices u and v is denoted as $w(uv)$, and the minimum length between u and v is called the distance (denoted as $d(u, v)$). For each vertex $v \in V$, let $l(v)$ be an upper bound of the distance $d(u_0, v)$, the Dijkstra's algorithm [31] has the following steps:

1. set $l(u_0) = 0$, $l(v) = \infty$ for all $v \neq u_0$; set u_0 as the root of search tree T; let

the vertex set $S_0 = \{u_0\}$ and $i = 0$;

2. for each $v \in \bar{S}_i$, replace $l(v)$ by $min\{l(v), l(u_i) + w(u_i v)\}$;
3. for each $v \in \bar{S}_i$ that connect to u_i, let v be a son of node u_i in T;
4. compute $\min_{v \in \bar{S}_i}\{l(v)\}$ and let u_{i+1} denote a vertex for which this minimum is attained, then set $S_{i+1} = S_i \bigcup \{u_{i+1}\}$;
5. if $u_{i+1} = v_0$, then stop; output $l(u_{i+1})$ as the distance from u_0 to v_0, and in T track the predecessors of u_{i+1} as the shortest path;
6. if $i = n - 1$ where n is the total number of vertices then stop; otherwise $i = i + 1$ go to step 2.

As pointed out in [15], the computation complexity of this algorithm is approximately $\frac{5}{2}n^2$.

5.5 Experimental Results

The technologies presented in the previous sections can be integrated into a comprehensive intelligent computer animation system. Given the CAD models of 3D objects, they can be transferred into attributed hypergraphs, on which automatic morphing and intelligent animation can be performed. In the animation, very limited user interactions are required, such as the number of frames per second, the displaying speed, the source and the target object or object status.

As in Section 4.6, a research prototype has been implemented with OpenGL and Motif on an SGI Indy Workstation (RISC4000 CPU) using IRIX 5.3.

5.5.1 3D Morphing

To show the efficacy of the proposed methodology, simulated and real world $2\frac{1}{2}$ or 3D data from various sources are tested. The experiments on 3D morphing are performed on (1) the real world range data of two human faces provided by National Research Council of Canada (NRC); (2) simulated 3D data of a cube and a pear; and (3) the 3D models of a number of crafts, also provided by NRC.

Morphing the Range Data of Human Faces

The data involved here are two range images, each of which is about 190×230 in size. The two original data sets are shown in Figure 5.6 (a) and (b) respectively.

The range images are first approximated by triangular meshes. The meshes of the source range data are sampled regularly so the result is a regular triangular mesh set. It has 625 vertices and 1152 triangles. However, the target range data are coarsened after the regular sampling. The coarsening is constrained by local geometrical features and the result is an irregular triangular mesh set, which consists of 379 vertices and 685 triangles. The top views of the two mesh sets are shown in Figure 5.6 (c) and (d).

Figure 5.7 (a) to (f) illustrate six frames of the morphing between the two faces, which give us a rough view of the gradual shape changes. The intensity images of the same two faces are also mapped to the $2\frac{1}{2}D$ meshes as textures. It can be seen that during the morphing, the AHR based algorithm interpolates not only the shape, but also the surface colors/textures of the objects.

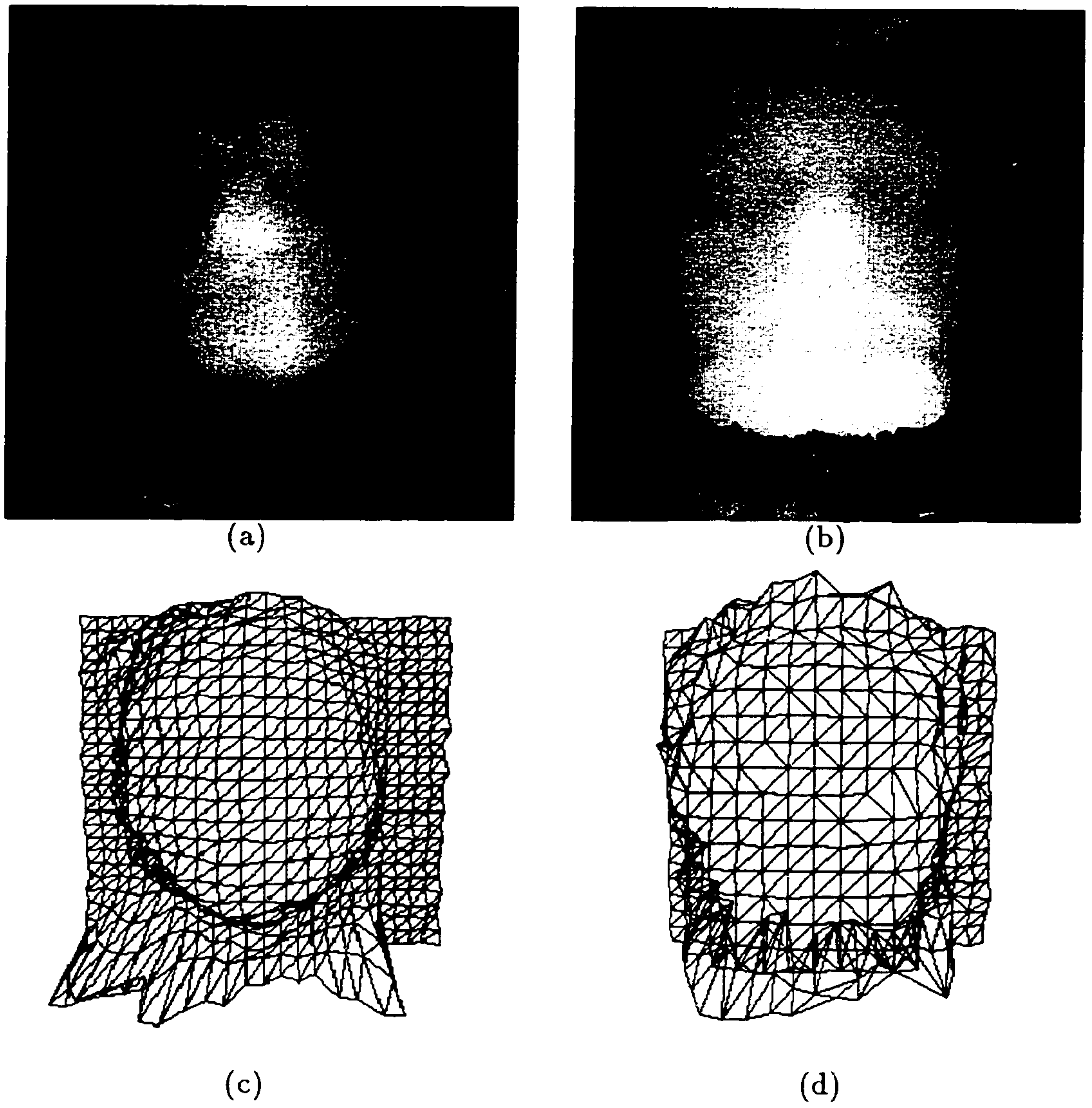


Figure 5.6: (a) the original range data of the source face; (b) the original range data of the target face; (c) the regular triangular meshes of the source range data; (d) the irregular triangular meshes of the target range data.

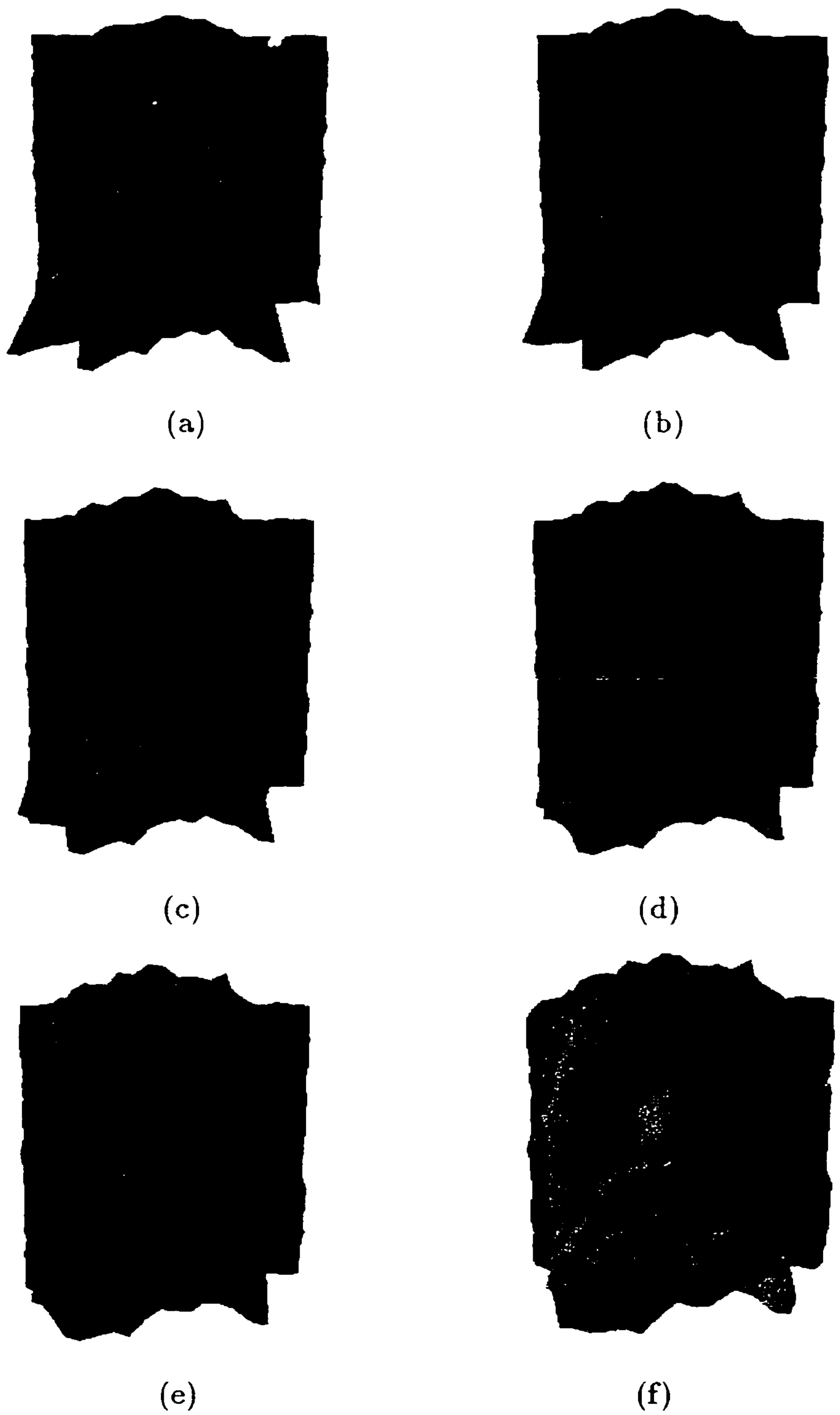


Figure 5.7: Six of the morphing frames between the range data of two human faces.

Morphing of Simple 3D Objects

This experiment is conducted on simulated data of simple 3D objects. It demonstrates the morphing from a simple cube to a pear, and then from the pear to an apple. The 3D data of the pear and the apple are downloaded from the Internet, originally in VRML format. They are converted into AHR's. Furthermore, mesh coarsening is performed on the data of the apple. The cube has 8 vertices and 12 triangles, the pear has 891 vertices and 1704 triangles, and the coarsened apple has 362 vertices and 691 triangles.

Figure 5.8 (a) through (i) demonstrate the morphing of the meshes, from a cube to a pear, and then from the pear to an apple. Figure 5.9 (a) through Figure 5.9 (i) illustrate the actual morphing frames displayed on the screen. It should be noted that the colors associated with the 3D objects are rendered with artificial lighting sources, which makes the presentation more realistic.

Morphing of Complex 3D Objects

Experiments on complex 3D data sensed from real world objects are conducted to test the system's efficacy in real world applications. With courtesy of National Research Council of Canada (NRC) by providing the 3D data set, the morphing is tested on the models of several crafts, including a vase, a totem pole from Aboriginal Canadian origins, a wooden toup and a toy duck. All original data from NRC are in VRML format and are then converted into AHR's. The data are considered complex since the meshes of each object consist of approximate ten thousand triangles and are fully mapped with color textures.

Figure 5.10 displays the result of the morphing between the meshes of the vase and the totem pole. Figure 5.11 show the result with texture morphing. Another

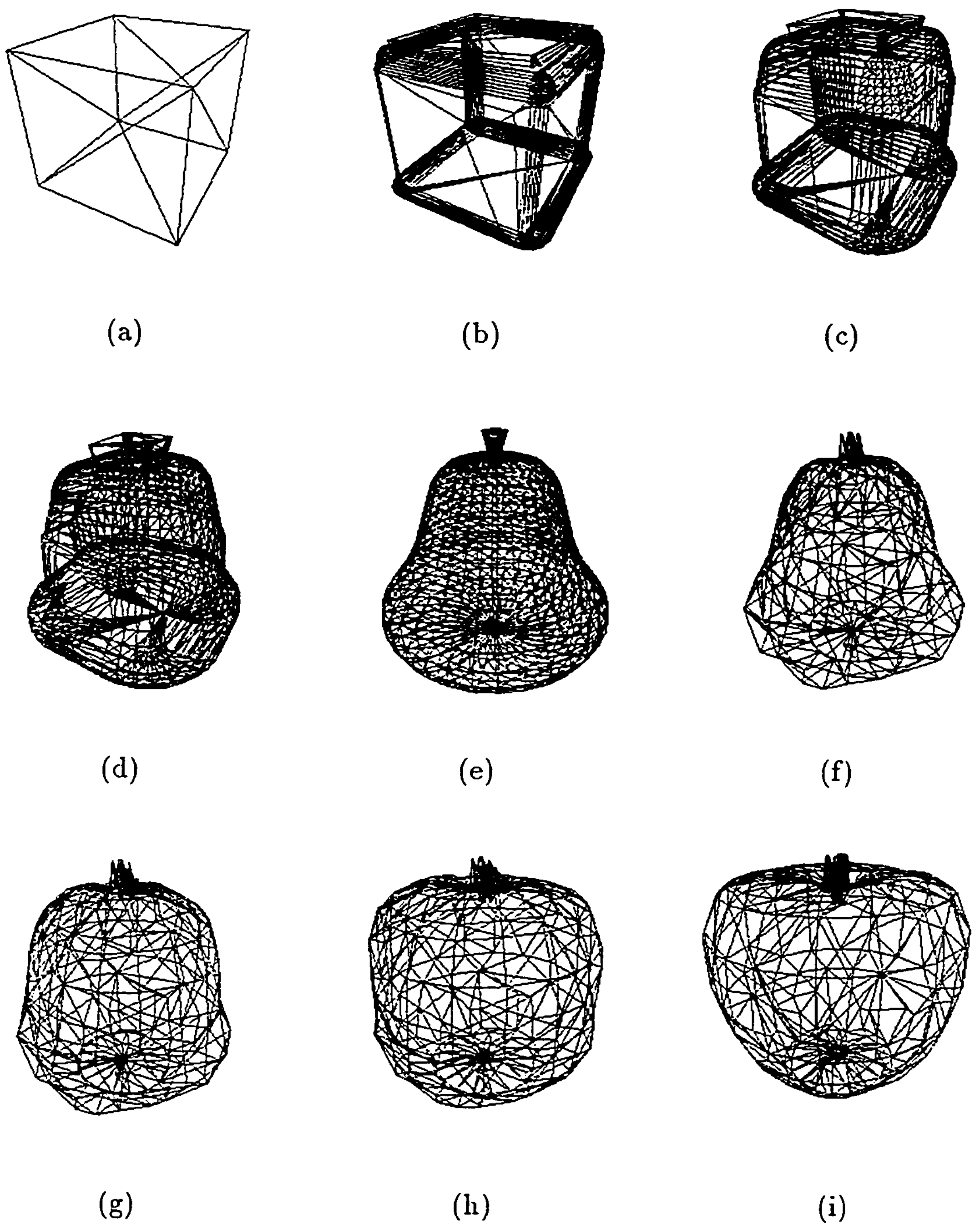


Figure 5.8: The morphing of the meshes, from (a) a cube to (e) a pear, and then from the pear to (i) an apple.

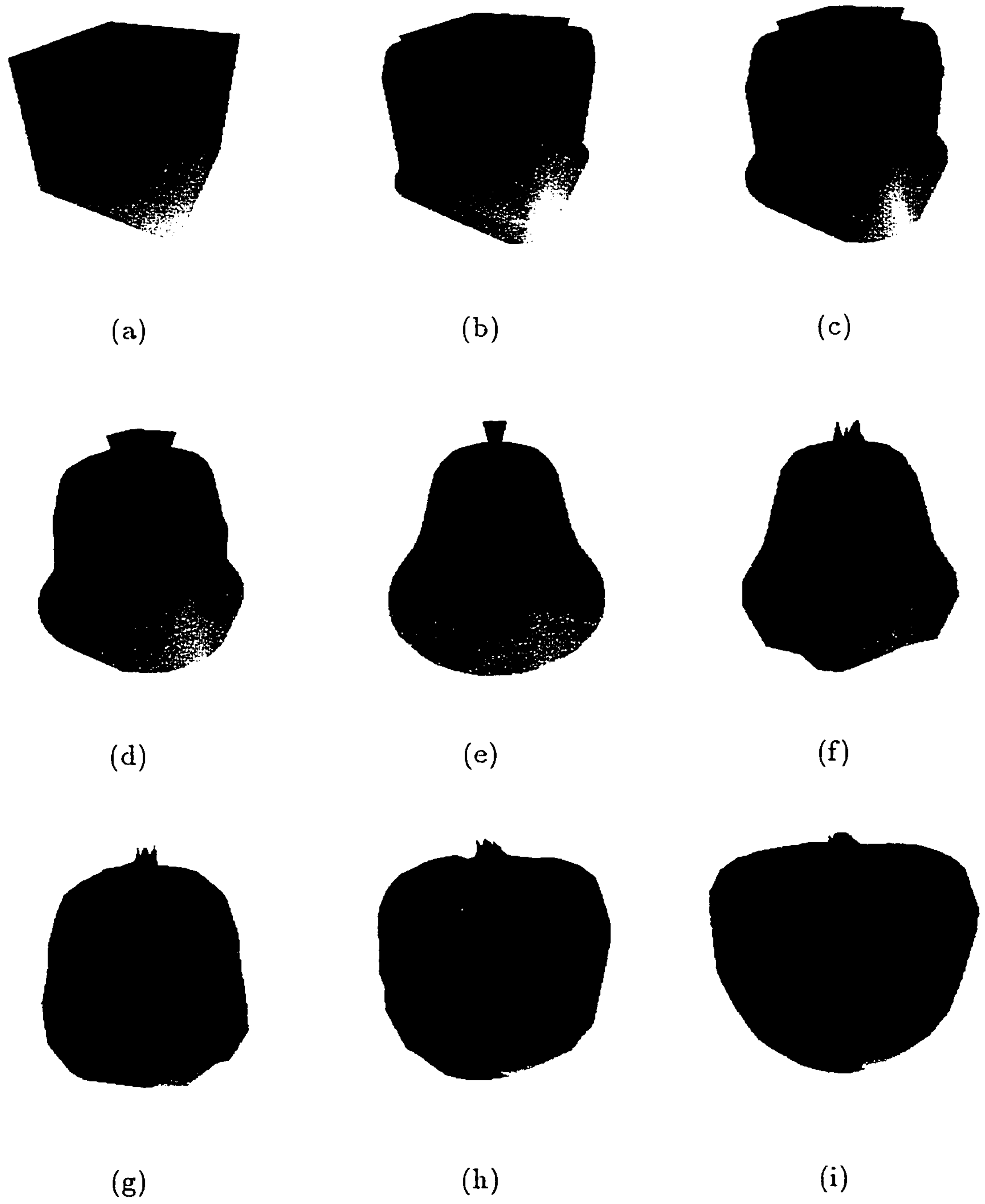


Figure 5.9: The morphing of simple 3D object, from (a) a cube to (e) a pear, and then from the pear to (i) an apple.

experiment was run on the data of the duck and the toup. The mesh morphing is shown in Figure 5.12 and the morphing with texture is illustrated in Figure 5.13. Figure 5.14 shows the same morphing with a virtual light source while the object rotates.

Discussion: Complexity of Optimal Subgraph Isomorphism

It has been stated in [100] that the complexity of the adopted optimal subgraph isomorphism algorithm for attributed graph is in the order of $O(k^3)$, where $k = n + m$ and n, m are the numbers of the vertices of the two graphs respectively. In the following experiments, pairs of DAH's (denoted as G_1 and G_2) with different number of nodes are matched. The matching cost measured in time (on the same SGI Indy workstation used in the morphing test) are tabulated as shown in Table 5.2. The numbers are also sketched in Figure 5.15 to show how the matching time grows versus k.

Theoretically, the matching cost is mainly decided by k. Thus as long as $k = m + n$ is the same, the matching costs with different combinations of m and n should not change significantly. However, experiments show that with the same k, the closer m and n are, the more time is used for matching. Moreover, the time used for the cases with $m = n$ can be twice as much as those of $\frac{m}{n} = 9$. The reason for such difference is that in the implementation, the heuristic search is initiated from the graph with less vertices, while the form of the heuristic function is dependent on the number of vertices in the graph. Recalling Equation 5.11, Equation 5.13 and Equation 5.14, we can see that $g(n)$ and $h(n)$ all have $\sum$'s to explore the whole or partial vertex set in the graph. Therefore, the complexity of the heuristic does depend on the order of the graph: the less vertices the graph has, the simpler the heuristic function would be.

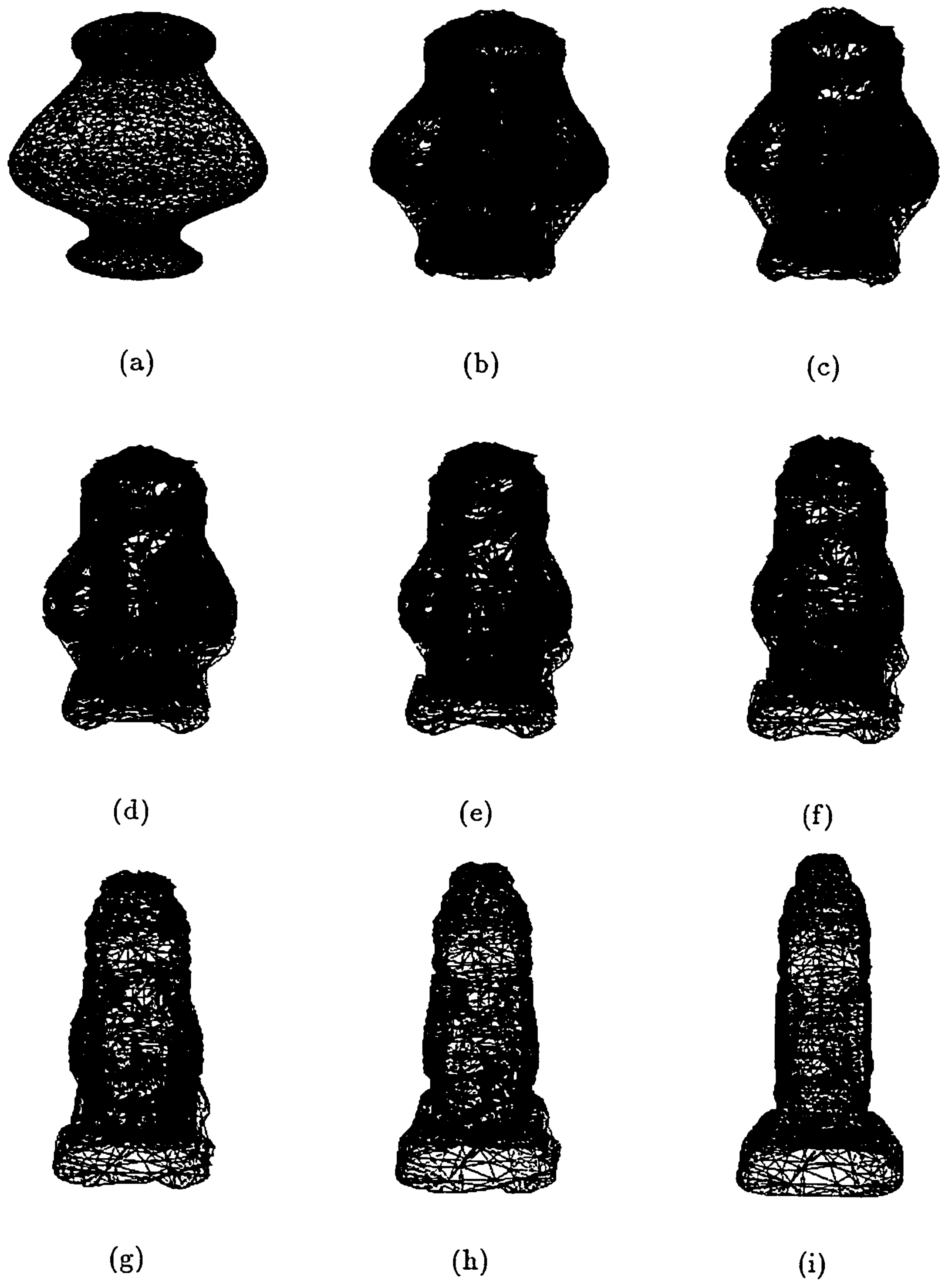
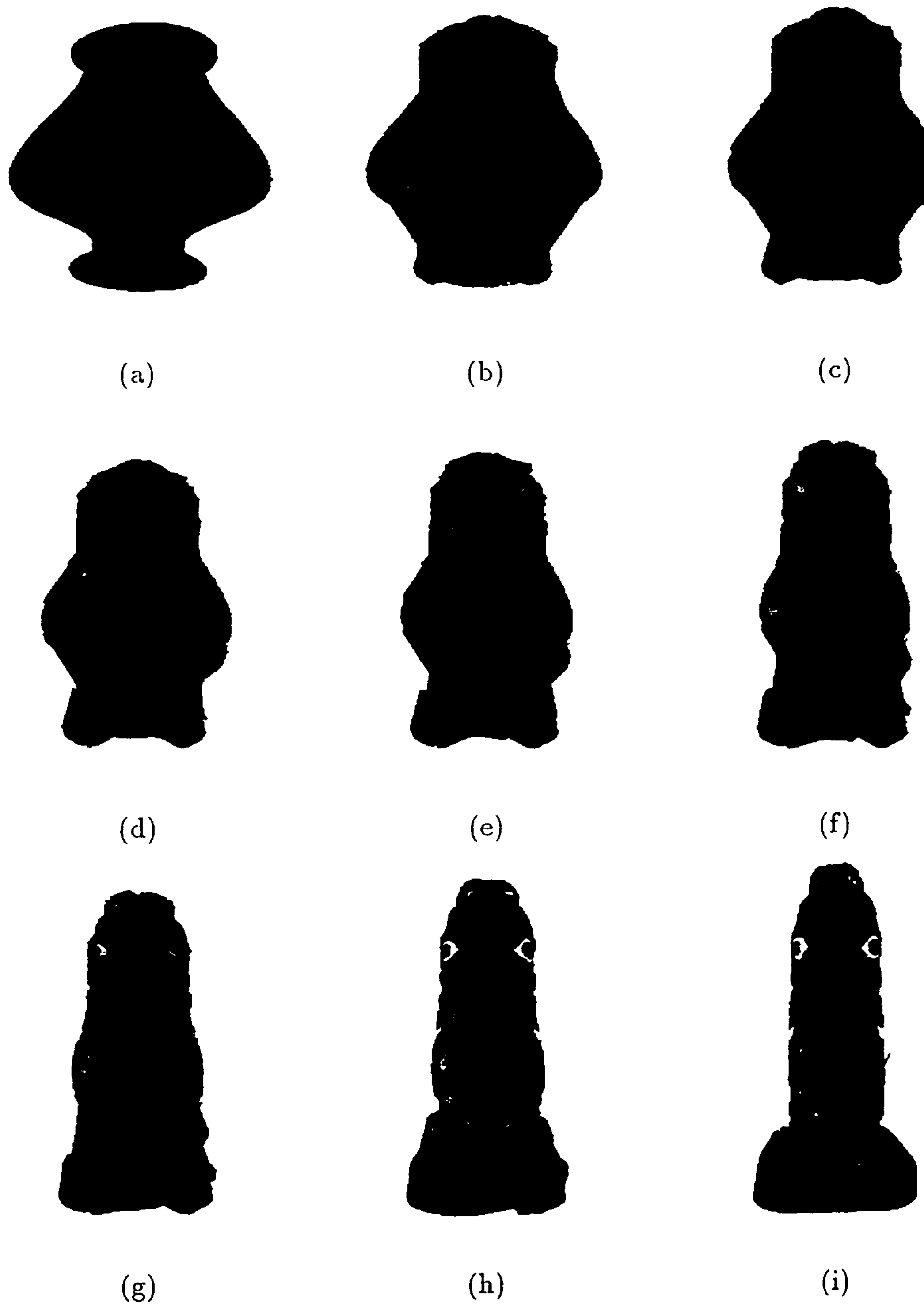


Figure 5.10: Nine of the morphing frames between the meshes of a vase and an Aboriginal Canadian's totem pole.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 5.11: The morphing frames of textured 3D data: a vase and an Aboriginal Canadian's totem pole.

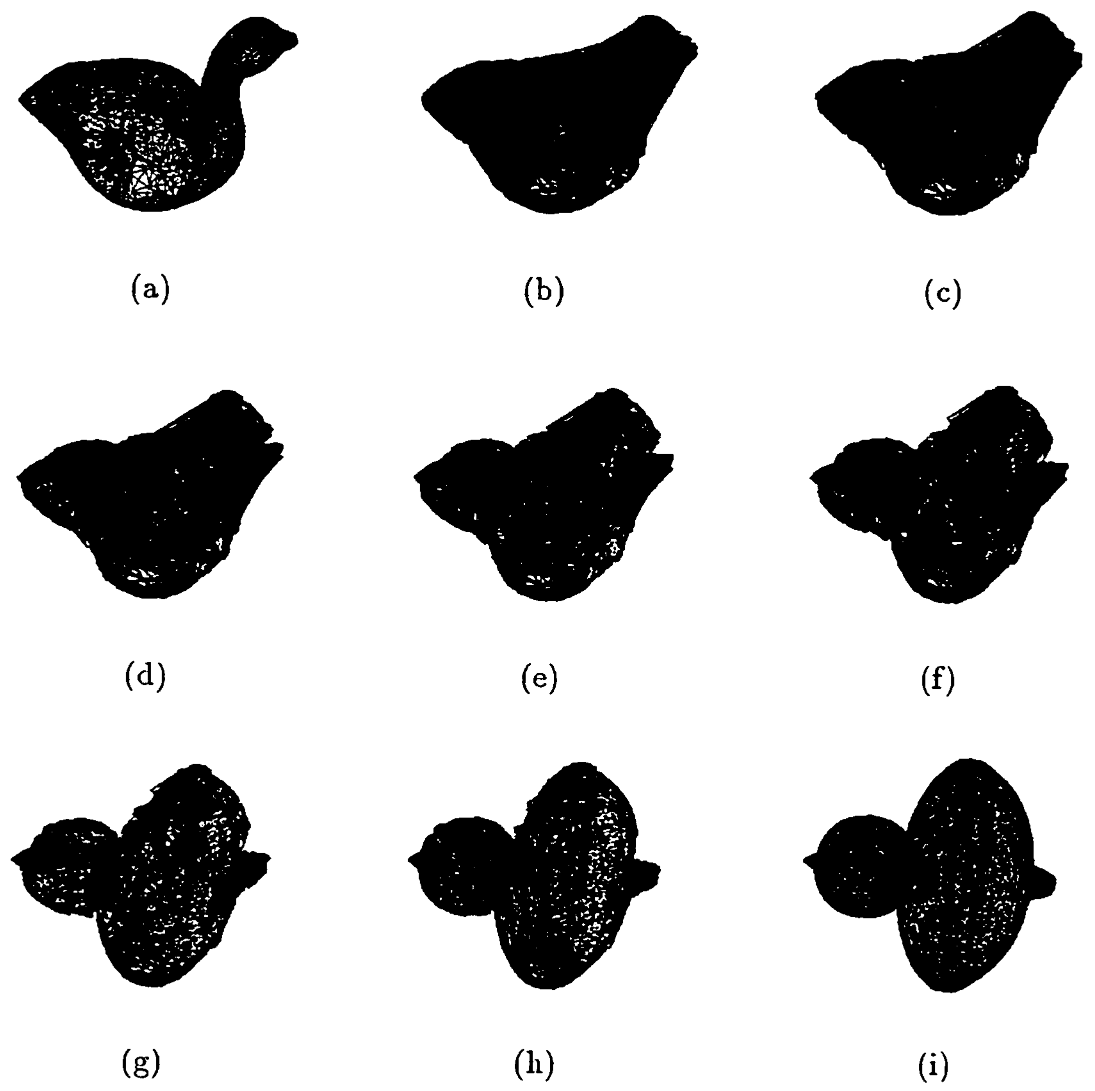


Figure 5.12: Nine of the morphing frames between the meshes of a duck and a toup.

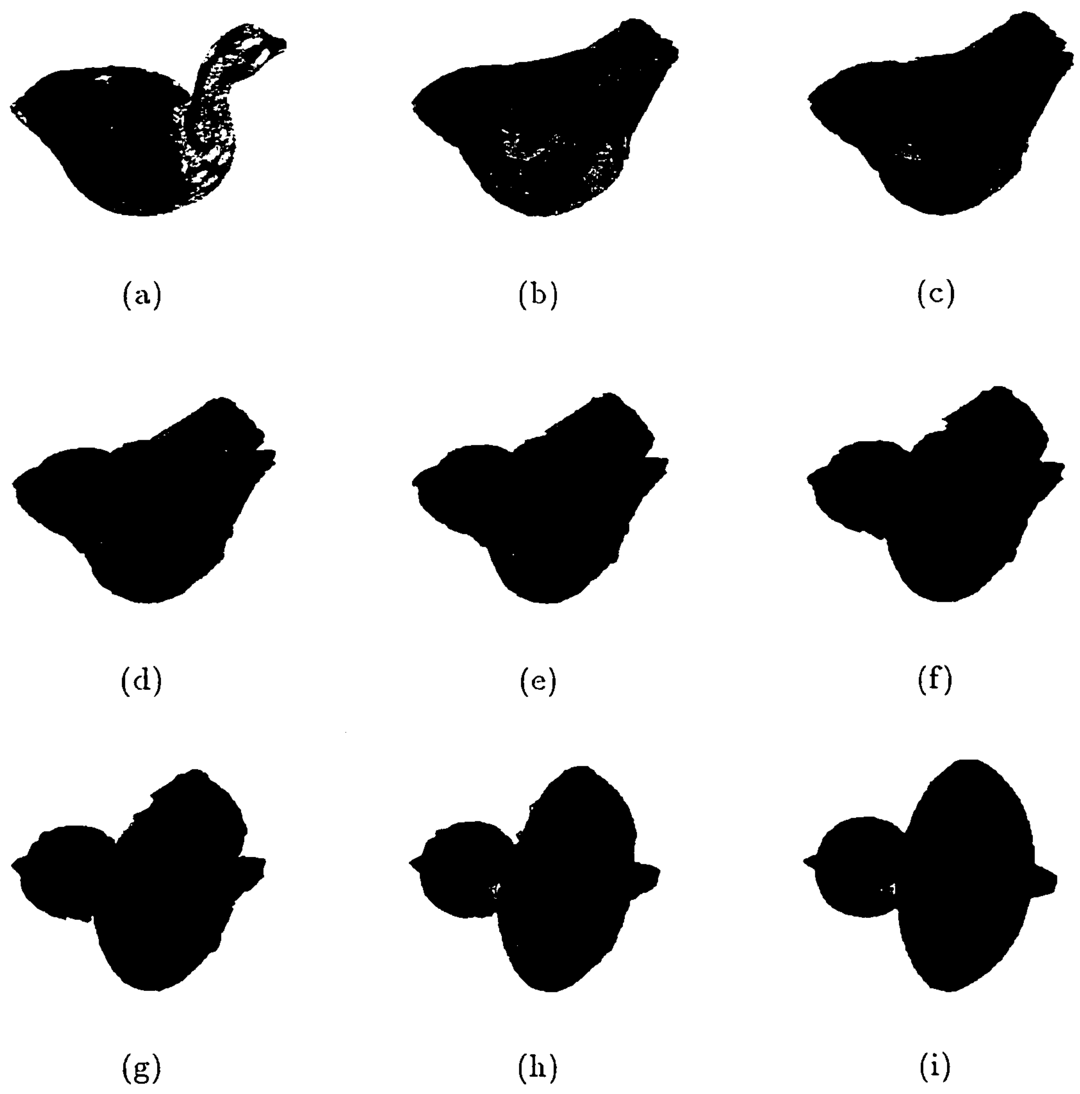


Figure 5.13: The morphing frames of textured 3D data: a toy duck and a toy toup.

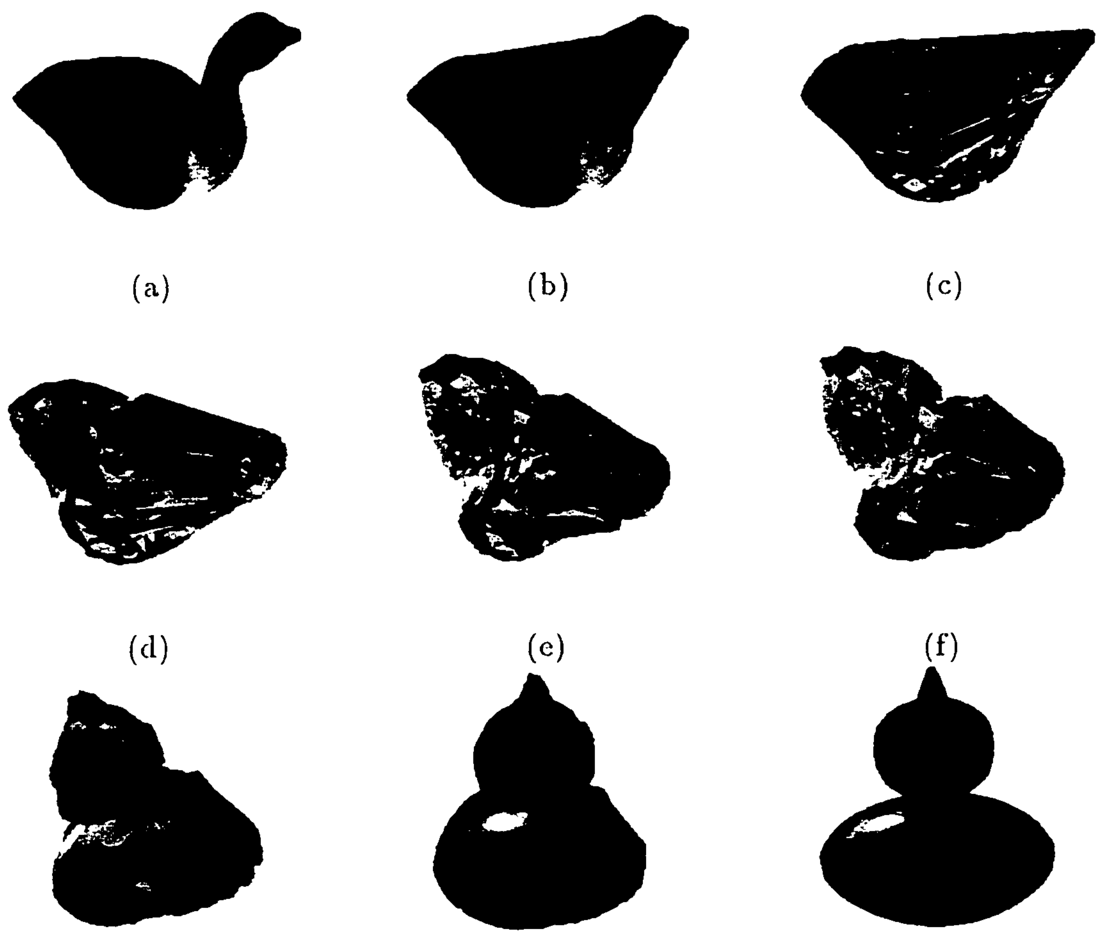

(a) (b) (c)
(d) (e) (f)
(g) (h) (i)

Figure 5.14: The morphing frames between a duck and a toup with rotation.

Table 5.2: The matching cost measured in the time used for attributed hypergraphs with different number of vertices.

m of G_1	n of G_2	k	matching cost (sec)
100	100	200	0.05
119	120	239	0.11
169	169	338	0.13
260	260	520	0.35
260	379	639	0.60
400	400	800	0.69
529	529	1058	1.22
676	676	1352	1.96
841	841	1682	3.05
900	900	1800	3.54
1024	1024	2048	4.52
1156	1156	2312	5.90
1444	1444	2888	9.22
1600	1600	3200	11.32

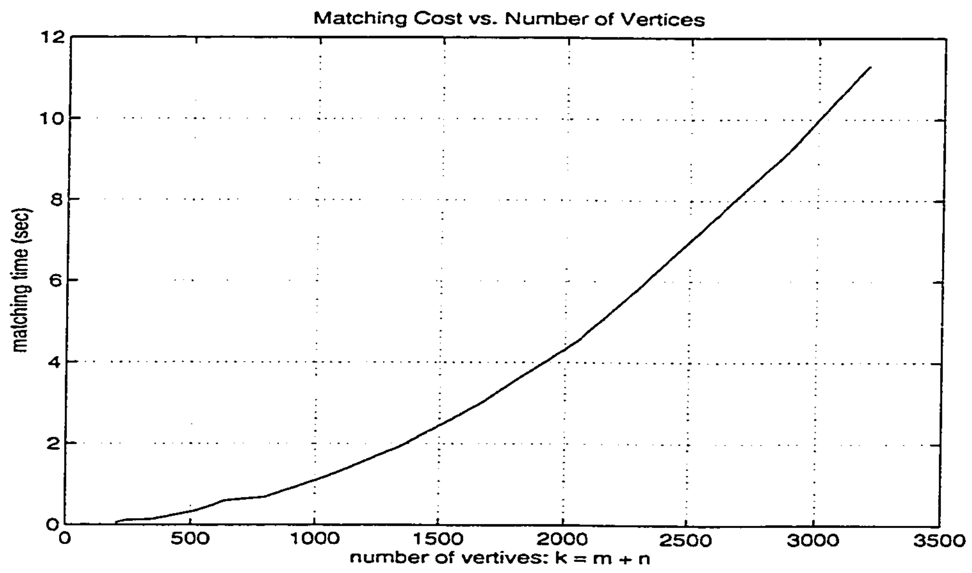


Figure 5.15: The matching cost (measured in time used by the second) versus k.

Table 5.3: The matching cost for attributed hypergraphs with different ratios of the vertex numbers.

$r = m/n$	matching cost (sec)	$r = m/n$	matching cost (sec)
0.9667	3.56	1.035	3.57
0.6602	3.51	1.515	3.50
0.4576	3.12	2.185	3.15
0.3100	2.93	3.225	2.98
0.2367	2.66	4.225	2.67
0.1786	2.35	5.600	2.38
0.1111	1.82	9.000	1.84
0.0625	1.41	16.00	1.40

The combinations of a set of specially designed DAH's are tested against the optimal subgraph isomorphism algorithm. All pairs of DAH's to be matched have the same $k = m + n = 1700$, but their ratios $r = \frac{m}{n}$ varies from 1 to 16. The results are shown in Table 5.3 and are plotted in Figure 5.16.

5.5.2 Intelligent Animation

The experiments of intelligent animation are carried on augmented 3D scene constructed in Section 4.6, which was shown in Figure 4.22 (e). In the AHR of the scene, four movable objects are defined:

1. the tetrahedron added in the scene, which consists of 4 triangles, is used as a simple virtual object in the augmented scene to test the path planning and animation methodology;
2. the robot model with more than 2000 triangles is an example of complex mobile object in the scene;

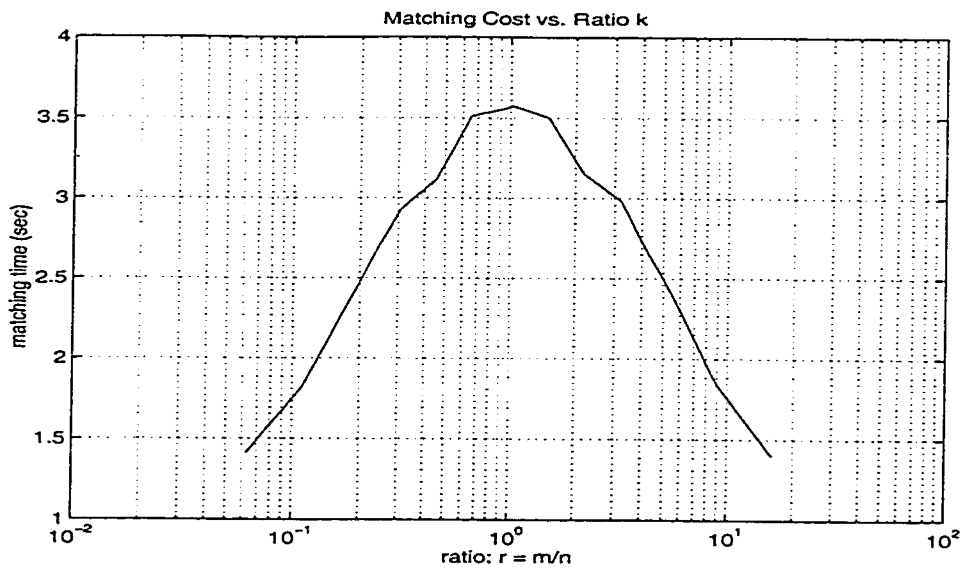


Figure 5.16: The matching cost (measured in time used by the second) versus different ratios between m and n while k is 1700 (the abscissa is the logarithm with base 10 of the ratio).

3. the red recycle bin acts as a movable object modeled from the real scene.

4. the door model is to test the specially constrained motion by animating it between its open and close positions;

The constraints on the trajectories of the first 3 objects are straightforward: the objects should always sit on the floor plane no matter where they go. With AHR, this can be implemented by simply limiting the search space within the hyperedge which represents the floor plane. For a procedural animation, the user is only required to specify: (1) the target's place on the floor plane, (2) the total time of the animation and (3) the number of frames per second (fps) desired. The optimal trajectory is generated automatically subject to the following intrinsic constraints:

- the layout of the floor plane;

- the object's movable direction;
- other objects presented in the scene (which may become the obstacles) for collision detection.

On the other hand, the animation of opening the door is quite different. It is somewhat pre-programmed as a knowledge-based animation: the fixed axle of the door is given by the user, and the motions of the door are governed by a procedural function. Suppose that the fixed axle of the door is chosen as the local z axle, the door (which is perpendicular to the $x - y$ plane) is at the local $x - z$ plane, the total time to open the door is assigned as T, the angle of door at opening position is $\Theta < 180^o$, then if on the door, a point P's original location at $t = 0$ is $P(0) = \{x_0, z_0, 0\}$, its location at $0 < t < T$ is $P(t) = \{x_0 \cos(\frac{t}{T}\Theta), x_0 \sin(\frac{t}{T}\Theta), z_0\}$.

Figure 5.17 gives the trajectory planning results for the objects in the scene illustrated at the representation level. The mobile robot is to walk outside through the door; the recycle box is to move to the back of the table; and the tetrahedron is to move to the front of the table. In the figure, the black links in the AHR represent the paths of the objects in animation. Figure 5.18 captures a frame of the animation.

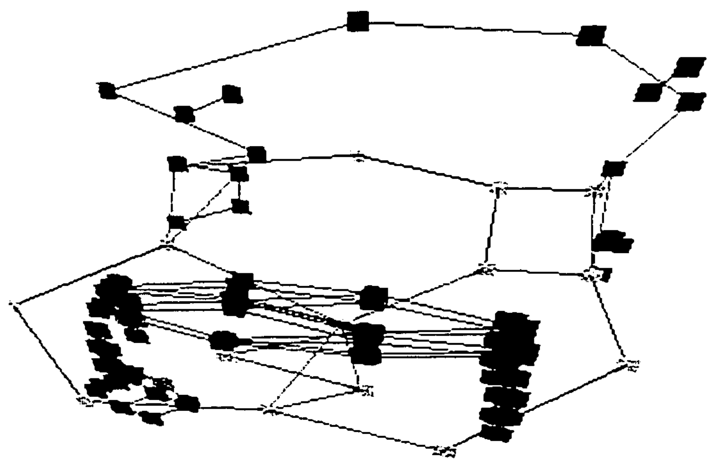


Figure 5.17: The trajectory planning results for the objects in the scene on the AHR.

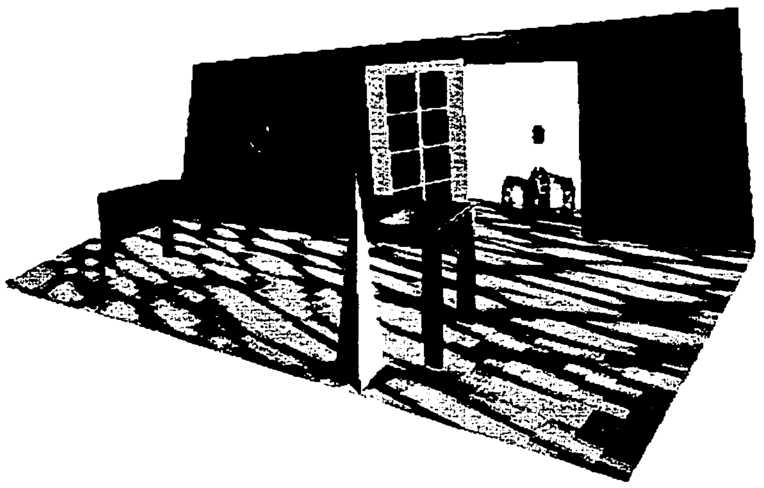


Figure 5.18: A frame captured during the animation in accordance with the planned paths.

Chapter 6

Conclusions And Future Work

6.1 Conclusions

This thesis presents a 3D object modeling system based on a unified attributed hypergraph representation (AHR). First, the mathematical aspect of the AHR based modeling methodology was investigated with the theoretical support from the category theory. Then the efficacy of the methodology was illustrated by preliminary experiments on 3D computer vision and animation applications.

The principal contributions of this work include:

- A mathematical modeling framework based on the category theory. It provides a theoretical support for the proposed AHR. In Chapter 3, the role of category theory in 3D object modeling was investigated from a mathematical perspective.

- A net-like structure called dynamic attributed hypergraph that supports fast operations on the entities of hypergraphs. The structure is layered, consisting of generic primitives, each of which is called a *node*.

- An algorithm of incremental model synthesis for 3D objects using CCD cameras. The synthesis procedure builds 3D object models based on the 2D features extracted from the images. The range data obtained by vertex triangulation are first organized into a face model and then to a triangular mesh. Finally, it is converted into an AHR.
- An AHR based algorithm to build an augmented reality by integrating virtual models and the models synthesized from real scenes. With the unified AHR framework, object's properties, such as surface colors and textures, are represented as attributes in a hypergraph, which makes it easy for us to perform the integrating and supplementing tasks.
- A distance measure for attributed hypergraphs. An optimal subgraph isomorphism algorithm based on the defined distance measure is applied to extract the descriptive parameters for non-rigid 3D motions. Substantial improvements to the efficiency of finding the optimal matching have been made through a heuristic function that prunes the search tree.
- An algorithm of 3D morphing based on AHR. It automatically generates the metamorphosis from a 3D shape to another one with continuous and controllable deformations and color/texture changes.

6.2 Future Work

The research listed above can be extended or enforced in several areas. It should be mentioned that the theories and the applications presented in this thesis are only implemented as research prototypes to demonstrate the efficacy of the methodology. A number of improvements in the implementations are no doubt possible. These may include:

- Currently, the vision based 3D modeling is not fully automatic due to the complexity and the noises presented in 2D image processing. Operator supervisions may be required to correct the errors. However, if equipped with a *structural lighting* based multiple-view vision system [48], the modeling process could be fully automated.
- The implementation of the functors that map various features to entities and/or operators in AHR can be further formalized. Many concepts and proofs in the category theory are believed to have physical meanings in AHR. They remain as open interests.
- The focus of the research was defined as machine vision based object modeling. Therefore, not much effort has been made on 3D rendering in morphing and animation. For developing deliverable packages as commercial products, professional rendering tools with state-of-art technologies have to be integrated with the modeling and manipulation engines.
- The implemented path planning algorithm is immature since the purpose is to show the possibility of AHR based intelligent animation. It may not handle tasks in complicate environments with sophisticated constraints. There are a handfuls of path/trajectory planning algorithms that can be ported to the proposed system.
- Many aspects of behavioral modeling and high level processes, such as the learning of motion rules and process abstraction, have not been implemented.

They are very important for many applications, such as emotion synthesis, human body animation and artificial life.

The proposed intelligent modeling methodology provides a very general representation scheme. Thus, its application is not limited to 3D object modeling. It may provide solutions for the following research projects with similar approaches:

- intelligent engineering visualization;
- physical process simulation;
- tele-conference and tele-operation;
- knowledge based automatic authoring/indexing in computer animation;
- procedural synthesis of phenomena;
- knowledge based artificial life.

All the above issues constitute the future challenges along the direction given by the research in this thesis.

Appendix A

Feature Based Mesh Coarsening

In this appendix, the feature based triangular mesh coarsening algorithm adopted from [35] is briefly described.

A Pascal-like pseudo-code of the algorithm's highest level decomposition is given in Figure A.1. The coarsening level index i is such that the triangulation $\mathcal{T}_{i+1}$ is coarser than $\mathcal{T}_i$. We use $T(i)$ to represent the data structure for $\mathcal{T}_i$.

```
PROCEDURE Coarsen($T(i)$, $\delta$, $\varepsilon$, VAR $T(i+1)$);
BEGIN {Coarsen}
        Extract_Significant_Edges($T(i)$, VAR $E(i)$);
        Group_Significant_Connected_Components($E(i)$, $\delta$, VAR $E_c(i)$);
        Select_Significant_Vertices($E_c(i)$, $\delta$, VAR $V_f(i)$, VAR $E_f(i)$);
        Grow_Selected_Vertices_Set($V_f(i)$, $\delta$, $\varepsilon$, VAR $V(i)$);
        Data_Dependent_Constrained_Triangulation($V(i)$, $E_f(i)$, VAR $T(i+1)$);
END {Coarsen};
```

Figure A.1: Pseudo-code for the coarsening algorithm.

The argument lists in the procedure calls are in the form: (*<input parameters>*, *VAR <output parameters>*). In the procedure calls in Figure A.1, they are defined as follows:

- δ is a parameter on feature distance measure defining the distance threshold, thus controlling the maximum size of the triangles;
- ε is an interpolation tolerance parameter controlling the accuracy of the resulting mesh in approximating the underlying surface;
- E is the set of all the edges in $\mathcal{T}$ which belong to significant edges;
- E_c is the reduction of E to those edges forming connected components;
- V_f is the set of prominent and/or important feature vertices in E_c;
- E_f is the set of edges connecting the vertices in V_f and approximating E_c;
- V is the augmented set of vertices ensuring a total coverage of the domain.

The procedure *Extract_Significant_Edges*() in Figure A.1 takes the set of all the edges in the initial triangulation $\mathcal{T}$ as input and produces the set of significant edges E as its output. The filtering decision about each edge in T is based on the analysis of the interface between the triangles sharing the edge. The procedure *Group_Significant_Connected_Components*() takes the set E as input and retains only those adjacent edges forming connected components with the variance of the feature values larger than the preset threshold δ. Its output is the reduced set of connected significant edges E_c. The set E_c is then fed as input to the procedure *Select_Significant_Vertices*() which identifies the vertices to preserve in the coarser triangulation. A vertex is preserved if it satisfies any of the following criteria:

- it is one of the end vertices of a connected component;

- it corresponds to "high" feature change rate on the connected component;
- it is an intermediate vertex *well-spaced* from its neighboring significant vertices according to the feature distance threshold δ.

The output of $Select\_Significant\_Vertices()$ is the set of those selected significant vertices V_f, together with a set of edges E_f, approximating the connected components. E_f need not be a proper subset of E. Nonetheless, edges in E_f can be considered as a coarse approximation of the features defined by edges in E.

In Figure A.1, procedure *Grow_Selected_Vertices_Set*() is the key to the efficiency of our approach. Figure A.2 shows its algorithmic description. The variables used in this procedure are defined as follows:

- Λ_o is the set of vertices still to be processed; we refer to it as the "open list";
- Λ_c is the set of vertices already processed; we refer to it as the "closed list";
- π is a cycle of connected edges in the original mesh whose edges enclose, but are not incident to v_k, the vertex of interest; and
- λ is the ordered list of vertices in the enclosing cycle π.

Ultimately, when all edges in π are expanded maximally, the maximal λ is used to determine the vertices to include in the coarser mesh. First, the vertices in λ which belong also to $\Lambda_o \cup \Lambda_c$ are identified. The criterion used for the selection of the remaining vertices is similar to the one used with the selected features. It is based on restricting the maximum feature distance between vertices to the value of δ. The set of these newly selected vertices Γ is added to the open list Λ_o for later consideration. Finally, v_k is moved from Λ_o to Λ_c, and the process repeated

```
PROCEDURE Grow_Selected_Vertices_Set($V_f$, $\delta$, $\varepsilon$, VAR $V$);
TYPE  vertex_set        $\Lambda_o$, $\Lambda_c$, $\Gamma$;
      vertex            $u_i$, $u_{i+1}$, $u_{i+2}$, $v_k$, $v'$, $w$;
      edge_list         $\pi$;
      vertex_list       $\lambda$;
      counter           $n$;
      error_measure     $Error(vertex_1, vertex_2)$;
BEGIN {Grow_Selected_Vertices_Set}
    $\Lambda_o \leftarrow V_f$;
    $\Lambda_c \leftarrow \phi$; {$\phi \equiv$ empty set}
    WHILE ($\Lambda_o \neq \phi$)
        Let $v_k \in \Lambda_o$;
        $\pi \leftarrow$ minimal enclosing cycle of $v_k$;
        $\lambda \leftarrow$ the ordered list of the $n$ vertices in $\pi$;
        Mark all vertices $v' \in [(\Lambda_o \cup \Lambda_c) \cap \lambda]$ , to be final in $\lambda$;
        WHILE [($u_i u_{i+1} \in \pi$) and ($u_i$ or $u_{i+1}$ not marked final in $\lambda$)]
            Let $w \leftarrow$ opposite vertex in the triangle containing $u_i u_{i+1}$
                (which is outside $\pi$);
            IF ($w \notin \lambda$)
                IF [($\|v_k w\| \leq \delta$) and ($Error(v_k, w) \leq \varepsilon$)]
                    Insert $w$ between $u_i$ and $u_{i+1}$ in $\lambda$;
                    $n \leftarrow n + 1$;
                    Replace $u_i u_{i+1}$ by $u_i w$ and $w u_{i+1}$ in $\pi$;
                    IF [$w \in (\Lambda_o \cup \Lambda_c)$]
                        Mark $w$ as final in $\lambda$;
                    ENDIF;
                ELSE
                    Mark $u_i$ and $u_{i+1}$ as final in $\lambda$;
                ENDIF;
            ELSE
                IF [($w = u_{i+2}$) and ($u_{i+1}$ not marked)]
                    Replace $u_i u_{i+1}$ and $u_{i+1} u_{i+2}$ by $u_i u_{i+2} = u_i w$ in $\pi$;
                    Delete $u_{i+1}$ from $\lambda$;
                    $n \leftarrow n - 1$;
                ENDIF;
            ENDIF;
        ENDWHILE;
        Find the set $\Gamma$ of vertices in $\lambda$ to include in $\Lambda_o$;
        $\Lambda_o \leftarrow \Lambda_o \setminus \{v_k\}$;
        $\Lambda_o \leftarrow \Lambda_o \cup \Gamma$;
        $\Lambda_c \leftarrow \Lambda_c \cup \{v_k\}$;
    ENDWHILE;
    $V \leftarrow \Lambda_c$;
END {Grow_Selected_Vertices_Set};
```

Figure A.2: The algorithm for generating the support vertices of the coarser mesh.

until Λ_o is eventually emptied. At that stage, Λ_c would have all the vertices to be included in the coarser mesh; it is returned as the output of the procedure.

Bibliography

[1] S. M. Ahmed and M. Borai, "Dual Camera Calibration for 3-D Machine Vision Metrology", *IEEE Trans. Instrumentation and Measurement*, 39(3):512-516, 1990.

[2] N. Alvertos, D. Brzakovic and R. C. Gonzalez, "Camera Geometries for Image Matching in 3-D Machine Vision", *IEEE Trans. PAMI*, 11(9):897-915, 1989.

[3] K. S. Arun, T. S. Huang and S. D. Blostein, "Least-square Fitting of Two 3D Point Sets", IEEE Trans. PAMI, 9(5), 1987.

[4] R. T. Azuma, "Tracking Requirements for Augmented Reality", *Communications of the ACM*, 36(7):50-51, 1993.

[5] R. T. Azuma, "A Survey of Augmented Reality", Teleoperators and Virtual Environments, Vol 6-4, August 1997, pp. 355-385.

[6] M. Bajura, H. Fuchs and R. Ohbuchi, "Merging Virtual Reality with the Real World: Seeing Ultrasound Imagery Within the Patient", In *Computer Graphics (Proc. of SIGGRAPH '92)*, 26(2):203-210, 1992.

[7] M. Bajura and U. Neumann, "Dynamic Registration Correction in Video-Based Augmented Reality Systems", *IEEE Comp. Graphics and Applications*, 15(5):52-60, 1995.

[8] S. T. Barnard and M. A. Fischler, "Computational Stereo", *ACM Computing Survey*, 14(4):553-572, 1982.

[9] A. Barr, "Superquadrics and Angle-preserving Transformations", *IEEE Computer Graphics Applications*, 18:21-30, 1981.

[10] A. Barr and A. Witkin, "Topics in Physically Based Modeling", *ACM SIGGRAPH '89*, Course Note 30, New York, 1989.

[11] A. Barr and Feigenbaum, (eds.) *The Handbook of Artificial Intelligence*, Vol. 4, Pitman Books, London, 1990.

[12] P. J. Besl, "Triangles as a Primary Representation", *NSF/ARPA Int. Workshop on 3D Object Representation for Computer Vision*, editor: M. Hebert em et. al., Springer, 1994.

[13] E. Bethel and S. Uselton, "Shape Distortion in Computer-Assisted Keyframe Animation", In *State of the Art in Computer Animation*, N. Magnenat-Thalmann and D. Thalmann eds., pp. 215-224, Springer-Verlag, New York, 1989.

[14] E. Boender, W. F. Bronsvoort and F. H. Post, "Finite-element mesh Generation from Constructive-solid-geometry models", *Computer Aided Design*, 26(5):379-392, 1994.

[15] J. A. Bondy & U. S. R. Murty, *Graph Theory with Applications*, NorthHolland, 1976.

[16] P. Borrel, "Simple Constrained Deformations for Geometric Modeling and Interactive Design" *ACM Trans. on Graphics*, 13(2):137-155, April 1994.

[17] D. Brutzman, M. Pesce, G. Bell, A. Dam and S. AbiEzzi, "VRML, Prelude and Future", *ACM SIGGRAPH '96*, pp. 489-490, New Orleans, August 1996.

[18] I. Bucur and A. Deleanu, *Introduction to the Theory of Categories and Functors*, John Wiley and Sons, Ltd., 1968.

[19] G. Burdea and P. Coiffet, *Virtual Reality Technology*, John Wiley and Sons, Inc., 1994.

[20] P. Buser and M. Imbert, *Vision*, translated by R. H. Kay, pp. 137-151, The MIT Press, 1992.

[21] D. Burr and J. Ross, "Visual Analysis during Motion", *Vision, Brian, and Cooperative Computation*, pp. 187-207, edited by M. A. Arbib and A. R. Hanson, The MIT Press, 1987.

[22] J. Canny, "A Computational Approach to Edge Detection", *IEEE Trans. PAMI*, (8)6:679-698, 1986.

[23] R. Carey and G. Bell, *The Annotaed VRML 97 Reference Manual*, Addison-Wesley Developers Press, 1997.

[24] P. Cavanagh and Y. G. Leclerc, "Shape From Shadows", *Journal of Experimental Psychology: Huamn Perception and Performance*, 15(1):3-27, 1989.

[25] Su-Shing Chen, "Structure form Motion without the Rigidity Assumption", *Proc. of the 3rd Workshop on Computer Vision: Representation and Control*, pp. 175-207, 1985.

[26] H. H. Chen, "Pose Determination from Line-to-plane Correspondences: Existence Condition and Closed Form Solution", *IEEE Trans. PAMI*, 13(6):530-541, 1991.

[27] Y. Chen and G. Mediono, "Surface Description of Complex Objects from Multiple Range Images", *Proc. IEEE Int. Conf. on CVPR*, pp. 153-158, 1994.

[28] T. S. Chua, K. H. Hay and W. N. Chin, "Optimization Constraint Model for Dynamic Animation", *Computer Graphics and Applications, Proc. of Pacific Graphics '93*, Vol 1, pp. 61-72, 1993.

[29] D. M. Fayek. "Vision-Guided 3D Object Model Synthesis from Range Data", M. A. Sc thesis, Dept. of Systems Design Eng., Univ. of Waterloo, 1996.

[30] P. Dierckx, *Curve and Surface Fitting with Splines*, Oxford University Press, 1993.

[31] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", *Numer. Math.*, 1:269-271, 1959.

[32] D. Drascic, J. Grodski, P. Milgram, K. Ruffo, P. Wong and S. Zhai, "ARGOS: A Display System for Augmented Reality", *Video Proc. of INTERCHI '93: Human Factors in Computing Systems*, pp. 24-29, April 1993.

[33] A. M. Earnshaw and A. K. C. Wong, "3-D Object Synthesis in a Monocular Imaging System", *Vision Interface '91*, Calgary, Alberta June 1991.

[34] R. E. Fayek and A. K. C. Wong, "Triangular Mesh Model for Natural Terrain", *Proc. of SPIE: Intelligent Robots and Computer Vision*, Oct. 1994.

[35] R. E. Fayek, "3D Surface Modeling Using Topographic Hierarchical Triangular Meshes", Ph.D Thesis, Systems Design Eng., University of Waterloo, Waterloo, Ontario, Canada, April 1996.

[36] M. Ferri, F. Mangili, and G. Viano, "Projective pose estimation of linear and quadratic primitives in monocular computer vision", *CVGIP*, 58(1):66-84, 1993.

[37] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: a Paradigm for Model Fitting Applications to Image Analysis and Automated Cartography", *ACM Communications*, 24:381-395, 1981.

[38] L. De Floriani, B., Falcidieno and C. Pienovi, "Delaunay-based Representation of Surfaces Defined over Arbitrary Shaped Domains", CVGIP, 32:127-140, 1985.

[39] L. De Floriani and E. Puppo, "Constrained Delaunay Triangulation for Multiresolution Surface Description", *Pattern Recognition*, pp. 566-569, 1988.

[40] S. J. Fortune, "A Sweepline Algorithm for Voronoi Diagrams", Algorithica, 2:153-174, 1987.

[41] Q. C. Gao and A.K.C. Wong, "A Curve Detection Approach Based on Perceptual Organization", *Pattern Recognition*, 26(7):1039-1046, August 1993.

[42] M. A. Garcia, "Fast Approximation of Range Images by Triangular Meshes Generated Through Adaptive Randomized Smapling", Proc. IEEE Int. Conf. on Robotics and Automation, 2:2043-2048, Nagoya, Japan, May 1995.

[43] M. A. Gigante, "Virtual Reality: Definitions, History and Applications" Chapter 1 in *Virtual Reality System*, eds., R. A. Earnshaw, M. A. Gigante and H. Jones, Academic Press, 1993.

[44] W. Grimson, T. Lozano-perez, W. Wells, G. Ettinger, S. White and R. Kikinis, "An Automatic Registration Method for Frameless Stereotaxy, Image Guided Surgery, and Enhanced Reality Visualization", *Proc. CVPR '94*, pp. 430-436, 1994.

[45] R. M. Haralick, "Monocular vision using inverse perspective projection geometry: Analytic relations", *Proceedings, CVPR '89*, pp. 370-378, 1989.

[46] T. He, S. Wang and A. Kaufman, "Wavelet-Based Volume Morphing", in D. Bergeron and A. Kaufman, eds., *Proc. of Visualization '94*, pp. 85-91, Oct. 1994.

[47] T. Hong, N. Magnenat-Thalmann and D. Thalmann, "A General Algorithm for 3-D Shape Interpolation in a Facet-Based Representation", *Proc. of Geraphics Interface '88*, pp. 229-235, 1988.

[48] T. Hong, "Flexible range finding applied to 3D surface synthesis: a pseudo-trinocular approach", Ph.D Thesis, Systems Design Eng., University of Waterloo, Waterloo, Ontario, Canada, 1992.

[49] R. Horaud and T. Skordas, "Stereo Correspondence Through Feature Grouping and Maximal Cliques", *IEEE Trans. PAMI*, 11(11):1168-1180, 1989.

[50] T. S. Huang, "Determining Three-Dimensional Motion and Structure from Two Perspective Views", in *Handbook of Pattern Recognition and Image Processing*, Chapter 14, Academic Press, 1986.

[51] J. F. Hughes, "Scheduled Fourier Volume Morphing", *Computer Graphics (Proc. SIGGRAPH'92)*, 26(2):43-46, July 1992.

[52] "http://www.isgtec.com/surgery.html", *Image-Guided Surgery.*

[53] C. P. Jerian and R. Jain, "Structure from Motion: A Critical Analysis of Methods", *IEEE Trans. SMC.* 21(3):572-588, 1991.

[54] BZ. Kamgar-Parsi, BR. Kamgar-Parsi and N. S. Netanyahu, "A Non-parametric Method for Fitting a Straight Line to a Noisy Image", *IEEE Trans. PAMI*, 11(9):998-1001, 1989.

[55] J. R. Kent, W. E. Carlson and R. E. Parent, "Shape transformation for polyhedral objects", In *Computer Graphics* (SIGGRAPH'92), 26(2):47-54, 1992.

[56] M. A. Khan and J. M. Vance, "A Mesh Reduction Approach to Parametric Surface Polygonization,", *1995 ASME Design Automation Conf. Proc.*, Boston, MA, Sept. 1995.

[57] C. L. Lawson, "Software for C^1 Surface Interpolation", in *Mathematical Software*, editor: J. R. Rise, pp. 161-194, Academic Press, New York, 1977.

[58] A. Lerios, C. D. Garfinkle and M. Levoy, "Feature-Based Volume Metamorphosis", *Proc. of SIGGRAPH '95*, pp. 449-456, Los Angeles, CA, August 6-11, 1995.

[59] C. W. Li, "Attributed Hypergraph Representation and Manipulation in Structural Pattern Recognition", Ph.D Thesis, Systems Design Eng., University of Waterloo, Waterloo, Ontario, Canada, 1996.

[60] L. Lienhardt, "Topological Models for Boundary Representation: A Comparison with N-dimensional Maps", Computer-Aided Design 23(1):59-82, 1991.

[61] H. C. Longuet-Higgins, "A Computer Algorithm for Reconstructing a Scene from Two Projections", *Nature*, 293:133-135, 1981.

[62] D. G. Lowe, "Perceptual Organization and Visual Recognition", Ph.D Thesis, Dept. of Computer Science, Stanford University, no. STAN-CS-84-1020, 1984.

[63] E. P. Lyvers, O. R. Mitchell, M. L. Akey and A. P. Reeves, "Subpixel Measurements Using a Moment-Based Edge Operator", *IEEE Trans. PAMI*, 11(12):1293-1309, 1989.

[64] D. Marr & Nishihara, "Representation and recognition of the spatial organization of three-dimensional shapes", *Proc. Royal Society of London B*, 200:269-294, 1978.

[65] J. H. McIntosh and K. M. Mucth, "Matching Straight Lines", *CVGIP*, 43:386-408, 1988.

[66] D. J. Meagher, "Octree encoding: a new technique for the representation, manipulation, and display of arbitrary three-dimensional objects by computer", *Technical Report, IPL-TR-80, 111*, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, NY, April 1982.

[67] M. C. Morrone and R. A. Owens, "Feature Detection from Local Energy", Pattern Recognition Letters, 6:303-313, 1987

[68] W. Niem and H. Broszio, "Mapping Texture From Multiple Camera Views Onto 3D Object Models for Computer Animation", *Proc. of the Int. Workshop on Stereoscopic and 3D Imaging*, Santorini, Greece, Sept. 1995.

[69] "http://www.sgi.com/Technology/Inventor.html", *Open Inventor.*

[70] M. Pavel, *Fundamentals of Pattern Recognition*, 2nd edition, Marcel Dekker Inc., 1993.

[71] T. Pavlidis, "Structural Description and Graph Grammars", *Pictorial Information Systems*, editors: S. K. Chang and K. S. Fu, Springer-Verlag, pp. 86-103, 1980.

[72] H. Peng, "Consistency, Accuracy and Robustness of Pose Determination in Computer Vision" M. A. Sc thesis, Dept. of Systems Design Eng., Univ. of Waterloo, 1995.

[73] J. M. Prager, "Extracting and Labeling Boundary Segments in Natural Scenes", *IEEE Trans. PAMI*, 10(2):167-192, 1988.

[74] S. Rippa, "Adaptive Approximation by Piecewise Linear Polynomials on Triangulations of Subsets of Scattered Data", *SIAM Journal on Scientific and Statistical Computing*, pp. 1123-1141, 1992.

[75] L.G. Roberts, "Machine perception of three-dimensional solids," in *Optical and Electro-Optical Information Processing*, J. Tippett, et al. (eds.), pp. 159-197, MIT Press, 1965.

[76] E. Rodabaugh, E. P. Klement & U. Hohle, *Applications of Category Theory to Fuzzy Subsets*, Kluwer Academic Publishers, 1992.

[77] L. L. Scarlatos and T. Pavlidis, "Hierarchical Triangulation Using Cartographic Coherence", *CVGIP: Graphic Models and Image Processing*, 54(2):147-161, 1992.

[78] , F. Schmitt, X. Chen and W. H. Du, "Geometric Modelling from Range Data", *Proc. Eurohgraphics'91*, pp. 317-328, Vienna, Austria, 1991.

[79] L. G. Shapiro, "Matching Three-dimensional Objects using a Relational Paradigm", *Pattern Recognition*, 17(4):385-405, 1984.

[80] A. C. Shaw, "Parsing of Graph-Representation Pictures", Journal of ACM, 17:453-481, 1970.

[81] D. Sims, "New Realities in Aircraft Design and Manufacture", *IEEE Computer Graphics and Applications*, 14(2):91, 1994.

[82] C. Smith, *An Elementary Treatise on Solid Geometry*, MacMillan, 1966.

[83] M. Soucy and D. Laurendeau, "A General Surface Approach to the Integration of a Set of Range Views", *IEEE Trans. PAMI*, 17(4):344-358, 1995.

[84] C. Terzides, "Transfomational Design", *Knowledge Aided Architectural Problem Solving and Design*, NSF Project #DMC-8609893, June 1989.

[85] D. Terzopoulos, J. Platt, A. Barr and K. Fleischer, "Elastically Deformable Models", *Computer Graphics*, 21(4):205-214, 1987.

[86] D. Terzopoulos and K. Fleischer, "Modeling inelastic deformation: Viscoelasticity, plasticity, fracture", *Computer Graphics*, 22(4):269-278, 1988.

[87] D. Terzopoulos and K. Fleischer, "Deformable Models", *Visual Computer*, 4:306-331, 1988.

[88] D. Terzopoulos and D. Metaxas, "Dynamic 3D models with local and global deformations: Deformable Superquadrics", *IEEE Trans. PAMI*, 13(7):703-714, 1991.

[89] D. Terzopoulos and K. Waters, "Analysis and Synthesis of Facial Image Sequences Using Physical and Anatomical Models", *IEEE Trans. PAMI*, 15(6):569-579, 1993.

[90] W. H. Tsai & K. S. Fu, "Error-correcting Isomorphism of Attributed Relational Graphs for Pattern Analysis", *IEEE Trans. SMC*, 9:757, 1979.

[91] R. Y. Tsai, "A Versatile Camera Calibration Technique for High-accuracy 3D Machine Vision Metrology Using Off-the-shelf TV Cameras and Lenses", *IEEE J. Robotics and Automation*, 3(4):323-344, 1987.

[92] G. Turk, "Re-Tiling Polygonal Surfaces" *Computer Graphics*, (*Proc. SIGGRAPH '92*), 26(2):55-64, 1992.

[93] F. Ulupinar and R. Nevatia, "Shape from Contour: Straight Homogeneous Generalized Cylinders and Constant Cross Section Generalized Cylinders", *IEEE Trans. PAMI*, 17(2):120-135, 1995.

[94] "http://vrml.sgi.com/moving-worlds/spec.DIS/index.html", *The Virtual Reality Modeling Language: ISO/IEC DIS 14772-1*, April 1997.

[95] B. Wanstall, "HUD on the Head for Combat Pilots", *Interactive*, 44, pp. 334-338, April 1989.

[96] J. Wernecke, *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor, Release 2*, Open Inventor Architecture Group, Addison-Wesley Publishing Company, 1994.

[97] A. Witkin and W. Welch, "Fast Animation and Control of Nonrigid Structures", *Computer Graphics*, 23(4):243-252, 1990.

[98] A. K. C. Wong and R. Salay, "An Algorithm for Constellation Matching", *Proc. of 8th Int. Conf. on Pattern Recognition*, pp. 546-554, Oct. 1986.

[99] A. K. C. Wong, M. Rioux and S. W. Lu, "Recognition and Shape Synthesis of 3-D Objects Based on Attributed Hypergraphs", *IEEE Trans. PAMI*, 11(3):279-290, 1989.

[100] A. K. C. Wong, M. L. You and S. C. Chan, "An Algorithm for Graph Optimal Monomorphism", *IEEE Trans. on S. M. C.*, 20(3):628-636, 1990.

[101] A. K. C. Wong, H. C. Shen and P. Wong, "Search Effective Multi-class Texture Classification", *Int. J. of PR and AI*, 4(4):527-552, 1990.

[102] A. K. C. Wong and W. Liu, "Hypergraph Representation for 3-D Object Model Synthesis and Scene Interpretation", *Proc. the 2nd Workshop on Sensor Fusion and Environment Modeling (ICAR)*, Oxford, U. K., 1991.

[103] A. K. C. Wong and B. A. McArthur, "Random Graph Representation for 3-D Object Models", *SPIE Milestone Series*, MS72:229-238, edited by H. Nasr, in Model-Based Vision, 1991.

[104] A. K. C. Wong, "3D Vision and Modeling", *Proc. of 2nd Int'l Conf. on Mechatronics & Machine Vision in Practice*, pp. 39-48, Hong Kong, 1995.

[105] A. K. C. Wong, P. Yu and X. Liang, "Elliptical Curve Detection Through Curve Data Grouping and Fitting", submitted to *CVGIP*.

[106] 2nd International Conference on the World Wide Web, Chicago, Oct. 1994.

[107] A. Yuille and D. Geiger, "Stereo and Controlled Movement", *Int. Journal Comp. Vision*, 4(2):141-152, 1990.

[108] S. W. Zuker, R. A. Hummel and A. Rosenfeld, "An Application of Relaxation Labeling to Line and Curve Enhancement", *IEEE Trans. Computers*, 26:394-403, 1977.